| | |
|---|---|
| CONTRACT NO.: | DAMD17-88-C-8018 |
| TITLE: | A TEST FIXTURE FOR SIMULATING HUMAN LIMB PHYSIOLOGY AND SOFT TISSUE BIOMECHANICS |
| PRINCIPAL INVESTIGATOR: | STEVEN M. FALK, MSE, PE |
| PI ADDRESS: | GMS ENGINEERING CORPORATION<br>8940-D ROUTE 108<br>COLUMBIA, MD 21045 |
| REPORT DATE: | APRIL 14, 1989 |
| TYPE OF REPORT: | FINAL REPORT |
| PREPARED FOR: | U.S. ARMY MEDICAL RESEARCH AND DEVELOPMENT COMMAND<br>FORT DETRICK<br>FREDERICK, MARYLAND 21701-5012 |

auth

Distribution ~~limited~~ to US Government Agencies only; Proprietary Information, April 14, 1989. Other requests for this document must be referred to Commander, US Army Medical Research and Development Command, ATTN: SGRD-RMA-RC, Fort Detrick, Frederick, MD 21701-5012.



*The views, opinions and/or findings contained in this report are those of the author and should not be construed as an official Department of the Army position, policy or decision unless so designated by other documentation.*

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION: Unclassified | 1b. RESTRICTIVE MARKINGS: N.A. Auth |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT: Distribution ~~Limited~~ To U.S. Gov't Agencies; Proprietary Information only April 14, 1989 |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION: GMS Engineering Corp. / 6b. OFFICE SYMBOL (If applicable) | 7a. NAME OF MONITORING ORGANIZATION |
| 6c. ADDRESS (City, State, and ZIP Code): 8940-D Route 108, Columbia, MD 21045 | 7b. ADDRESS (City, State, and ZIP Code) |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION: U.S. Army Medical R&D Command / 8b. OFFICE SYMBOL (If applicable): SGRD-RMA-RC | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: DAMD17-88-C-8018 |
| 8c. ADDRESS (City, State, and ZIP Code): Fort Detrick, Frederick, MD 21701-5012 | 10. SOURCE OF FUNDING NUMBERS |

| PROGRAM ELEMENT NO. | PROJECT NO. 3E1- | TASK NO. | WORK UNIT ACCESSION NO. |
|---|---|---|---|
| 62787A | 62787A878 | AF | 146 |

11. TITLE (Include Security Classification)

A Test Fixture For Simulating Human Limb Physiology And Soft Tissue Biomechani

12. PERSONAL AUTHOR(S)
S. M. Falk, MSE, PE

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Final | FROM 3/88 TO 4/89 | April 14, 1989 | 100 |

16. SUPPLEMENTARY NOTATION

Reprints

| 17. COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB-GROUP |
| 06 | 02 | |
| 06 | 11 | |

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Vital Signs Monitor; Test Jig; Artifical Arm; RA II; (K-)

19. Evaluation of the reliability and accuracy of fieldable vital signs monitoring devices has been a difficult problem. In the civilian sector, the problems are partially circumvented by limiting the operational and environmental requirements. Military vital signs monitors must perform reliably across extreme temperatures, over unique garments, in high noise and vibration environments and other hostile surrounds. A major obstacle in evaluating potential fieldable monitoring devices has been the unavailability of a reliable standard test-jig which accurately simulates the physiology of the human limb with respect to its vascular supply, blood pressure and heart rate. This project was undertaken in order to demostrate the feasibility of creating such a device. The first task of this project was to first obtain a thorough understanding of how the human limb develops the physiologic signals that the various monitoring instruments detect and interpret as blood pressure and pulse rate. This was accomplished by formulating a theory which predicts limb volume changes in response to the pressurization of occluding blood pressure cuffs and the resulting changes in blood flow within the underlying arteries and veins. On the basis of this theory, a model arm was constructed so as to incorporate mechanical analogues of collapsible tubes. Simulated blood flow and pressure were driven by a programmable gear pump. Auscultatory, oscillometric, and statometric methods were used to evaluate the model. The success of the project is indicated by the fact that all of these techniques yield reasonable values of simulated pressures and pulse rates. The most important evidence for the success of this effort is that the model does not have to be "informed" about the specific technique that any available instrument utilizes. A major accomplishment is the development and successful implementation of a single theory which explains the physical and physiological phenomena associated with all three major noninvasive blood pressure measurement techniques. A number of problems and suggestions for future improvements in the device have been identified.

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21. ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS | Unclassified |

| 22a. NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE (Include Area Code) | 22c. OFFICE SYMBOL |
|---|---|---|
| Mrs. Virginia M. Miller | 301/663-7325 | SGRD-RMI-S |

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

# FOREWORD

Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the U.S. Army.

Where copyrighted material is quoted, permission has been obtained to use such material.

Where material from documents designated for limited distribution is quoted, permission has been obtained to use the material.

Citations of commercial organizations and trade names in this report do not constitute an official Department of the Army endorsement or approval of the products or services of these organizations.

For the protection of human subjects, the investigator has adhered to policies of applicable Federal Law 45CFR46.

PI Signature 4-14-89 Date

| Accession For | |
|---|---|
| NTIS GRA&I | ☐ |
| DTIC TAB | ☑ |
| Unannounced | ☐ |
| Justification | |
| By | |
| Distribution/ | |
| Availability Codes | |
| Dist | Avail and/or Special |
| B-3 | |





"Use or disclosure of report data is subject to the restriction on the Title page of this Final Report".

## TABLE OF CONTENTS

## LIST OF FIGURES

# 1 INTRODUCTION

Empirical evaluation of the reliability, accuracy, and reproducibility of vital signs monitoring (VSM) devices has historically been an intractable problem. In the civilian sector, the problem has been partially circumvented by drastically limiting the prescribed operational environment requirements.

In the case of military VSM devices, it is impossible to drastically constrain the operational environment requirements. Military vital signs monitors must perform reliably in conventional and NBC warfare situations, across extreme temperatures, over unique garments, in high noise and vibration environments, over a wide range of altitudes, and in contaminated surroundings. These severe operational requirements have prompted the Army, Air Force, and Navy to invest many millions of dollars over the past six years in an effort to develop a system capable of supporting battlefield casualties. As a consequence, the unique requirements of the DoD and the extensive funding of this research and development effort have resulted in significant advances in the state-of-the-art of vital signs monitoring instrumentation.

While these advances greatly surpass commercially available technology, they have not been accompanied by comparable advances in requisite test and evaluation methodology and instrumentation. As a result, we now find ourselves in the disheartening position of having sophisticated, high performance devices, but grossly inadequate methodology and instrumentation for credible evaluation of device reliability, accuracy, and reproducibility.

The purpose of this research and development effort is the development of testing and evaluation methodology and instrumentation which will permit the credible evaluation of VSM devices in both the advanced development phase and in the production quality control phase.

# 2 BACKGROUND

The task of evaluating medical devices for military deployment includes the determination of device survivability and functional performance. The issue of VSM device survivability can be adequately determined with existing test and evaluation (T&E) methodology and instrumentation. These procedures are documented in various military standards documents (e.g., MIL-STD 810D, 461, 462, etc.). In contrast to this, the issue of VSM device functional performance is not covered by any applicable military T&E standards; thus, guidance is sought from civilian/commercial

standards.

At the present time the Food and Drug Administration (FDA) has not promulgated performance standards for vital signs monitors that determine blood pressure and heart rate. The FDA's principal function regarding medical devices is to verify safety and manufacturers' claims regarding performance. The FDA informally employs a voluntary industry performance standard promulgated by the Association for the Advancement of Medical Instrumentation (AAMI SP-10) for the assessment of electronic blood pressure monitors. With regard to heart rate, we are not aware of any FDA or non-FDA performance standards.

AAMI SP-10 requires clinical evaluation of electronic blood pressure monitors. For each functional performance test condition, a minimum of fifty **noninvasively** instrumented human subjects or fifteen **invasively** instrumented human subjects are required.

Since commercial civilian devices normally have only ONE test condition (room temperature, bare arm, NO noise or vibration, near sea-level altitude), the experimental risk involved with the use of fifteen invasively instrumented human subjects (fluoroscopically instrumented with an intra-arterial catheter) is relatively small. However, if many different combinations of test conditions are required (as is the case with military requirements), the associated risk to human volunteers rapidly increases to a level at which invasive testing is not justifiable.

The alternative to invasive testing is to noninvasively instrument the human test subjects. The AAMI standard requires a minimum of fifty such human subjects (having a wide range of blood pressures) and requires that the device performance be determined by comparison to the auscultatory method (detection of Korotkoff sounds via a stethoscope by a trained technician). In addition to the fact that multiple test conditions require multiple measurements on the 50+ human subjects, using the auscultatory method as a standard presents a MAJOR problem for testing under military operational conditions. The auscultatory method DOES NOT work in high noise and vibration environments; it is very unreliable when obtained through heavy garments (such as MOPP gear), and is biased by the subjectivity of the measurement obtained by the technician[1]. While this subjectivity error can be controlled under normal conditions, testing under adverse conditions (e.g., extreme heat or cold) will significantly confound the measurement, necessitating that the experimental design include many more measurements than normally than are normally required.

Furthermore, the blood pressure and heart rate are very labile physiological parameters - they can change dramatically in very short time periods. Reasons for such changes include physical, physiological, and psychological stress. For example, systolic blood pressure has been shown to change as much as 50 mmHg in as little as 30 seconds[2]. Heart rate can decelerate to zero in as little as one beat or can increase as much as 50% in a few seconds. All techniques measure a physiological parameter at a specific point in time. Unless one is certain that blood pressure and heart rate are

stable, the results from a single measurement become invalid in a very short time period and repeated measurements are essential. Furthermore, during rapidly changing blood pressure and heart rate conditions, both manual and automatic measurements yield erroneous results. That is, the measurement does not yield either the initial value nor the final value, but some intermediate value of questionable validity.

Multiple measurements on many human subjects takes a considerable amount of time. Each time a counter-pressure cuff is inflated to occlude blood flow to an extremity, it causes ischemia and disturbs the normal physiology of the extremity. This is a real limitation of all noninvasive blood pressure measurement techniques. Intermittent occlusion, separated by reasonable time intervals, permits restoration of normal physiology to the extremity. However, rapid repetition of occlusion by a cuff will, as time progresses, become more and more like continuous occlusion. This is contra-indicated not only because of the adverse effects on the extremity, but also because it attenuates the physical phenomena normally employed in the measurement of blood pressure. While two or three measurements in rapid succession should not create problems, monitoring over a period of an hour or so should be done at about five minute intervals and monitoring for longer time periods should be done using longer time intervals between measurements. In this way, the effect of the previous measurement does not affect the results of the subsequent measurement.

The requirement for multiple human subjects results in the necessity for multiple cuff applications. Proper selection and application of the cuff is important for accurate determination of blood pressure. In both the oscillometric and statometric blood pressure measurement technique, the cuff is the ultimate transducer, without which a measurement cannot be achieved. In automated versions of the auscultatory technique, the microphone is normally located within the cuff, once again making proper cuff placement essential. Proper selection of cuff size, relative to the diameter of the extremity will prevent false or cuff hypertension[3]. Cuff hypertension is an erroneously high blood pressure reading that occurs because the pressure in the cuff is not completely transmitted to the underlying artery. This normally occurs when the cuff width is small compared to the circumference of the extremity. Cuff placement is also important in automated blood pressure measurement. In a device using the auscultatory technique, placement of the microphone over an artery maximizes signal detection and minimizes erroneous readings. In a device that uses the statometric technique, the distal sensing cuff acts analogous to a "microphone". The best placement requires that the middle of the air bladder be located over an artery, but without contacting the proximal occluding cuff (e.g. over the radial artery of the arm immediately below the elbow).

All instruments are subject to noise/vibration induced errors and "noise immunity" is always a matter of degree. While it is normally impossible to prevent vehicle-related noise/vibration during patient transport, this is not usually the case with patient-related noise/vibration Patient motion

represents a significant impediment to the measurement of blood pressure; this is obvious for manual phenomena that either mimic or interfere with the signals that automated devices use for determining blood pressure. Gross movement of the whole arm or leg can create "false" pressures in the counter-pressure cuff, leading to a blood pressure that is either higher or lower than the correct reading. Flexion, extension, or rotation of the hand or foot, especially in a rhythmic fashion, can create signals identical to those used by the oscillometric and statometric techniques. The ultimate defense against patient motion artifacts is to prevent patient voluntary motion. The penultimate defense is to observe the patient's extremity during the measurement and exercise caution in interpreting the results if significant patient motion was observed during the measurement.

One must also be aware of the pressure difference due to the position of the extremity. The vertical distance between the heart and the blood pressure sensor causes a systematic error in the blood pressure measurement[4]. For every five inches of vertical distance, the pressure error exceeds nine millimeters of mercury.

# 3 APPROACH

In order to fabricate a test fixture for vital signs monitors, we must completely understand the physics and physiology of each technique for noninvasive blood pressure measurement - auscultatory, oscillometric, and statometric.

## 3.1 General

In order to measure arterial pressure noninvasively, one must apply a blood pressure cuff around the arm. The cuff pressure is transmitted through the tissue and acts on the outside surface of the artery. If the external pressure acting on the artery is greater than the internal arterial pressure, the artery collapses, interrupting flow in the artery. Various techniques detect this event, reporting the cuff pressure required for the zero flow state as systolic pressure. These techniques also measure or estimate diastolic and mean pressures.

## 3.2 Auscultatory

The auscultatory method of measuring blood pressure consists of listening for Korotkoff sounds as the cuff is deflated from above systolic pressure to below diastolic pressure. There is disagreement as to the origin of these sounds[5]. Korotkoff[6] believed that the sounds were caused by the arterial walls "slapping" against each other as pulses of blood passed through the partially occluded artery. Gittings[7] believed that the sounds were caused

by sudden distention of the compressed, partially occluded artery. Other theories include a mechanism involving dynamic instability of a viscoelastic system of tissue[8]. Conrad et al believe that these sounds are caused by fluid turbulence[9]. No conclusive scientific evidence has yet been obtained to prove or disprove any of these theories.

In our attempt to model the human limb, we used a combination of Korotkoff's theory and that of Conrad et al. The vessel which we have designed permits the "slapping" of the artery as it closes upon partial occlusion. In addition, a microphone, which is fluidically coupled to the arterial line distal to the occluding cuff, is used to pick up the sound for further amplification. Thus, any sounds generated by turbulence or "slapping" can be sensed at this position.

### 3.3 BFS Theory

The BFS Theory[a] (Blaumanis, Falk, and Samaras) describes the principles of the noninvasive measurement of blood pressure. Researchers in the field of venous occlusion plethysmography noticed that when an occluding cuff was applied to the arm and the cuff pressure was held at a pressure above venous occlusion pressure, the distally measured intravenous pressure, became equal to the cuff pressure[10]. We repeated these experiments and found that the intravenous pressure equals that of the cuff from a pressure just above venous occlusion until mean arterial pressure (MAP) is reached.

The explanation is as follows. Consider the limb, distal to the occluding cuff, as a control volume. Normally, on average, the arterial inflow equals the venous outflow. The entire pressure drop (arterial-venous) occurs at the level of the microcirculation, the arterioles and capillaries. Now, if the cuff is inflated to just above venous occlusion pressure (say 10 mmHg) the outflow will cease momentarily. As a result of the diminished flow, the total pressure drop is also diminished, since the latter is due primarily to the viscous frictional losses in the microcirculation. Thus, the reduced flow, due to outflow obstruction, results in an increased pressure within the down-stream elements of the vasculature, the venules and veins. The incipient increase in venous pressure under the occluding cuff will, therefore, tend to open the collapsed veins and flow will resume. At this point a dynamic equilibrium is established where venous pressure is, on average, equal to cuff pressure. If we now inflate the occlusion cuff to higher and higher pressures new equilibrium points will be established such that venous pressure rises to the level of the cuff pressure. Obviously a point must be reached where venous pressure will equal arterial pressure and flow will be zero. At this

---

[a] An article is being prepared for publication in the open scientific literature.

point an interesting question arises: when net flow is just reduced to zero by venous occlusion, what is the venous pressure and therefore the cuff pressure? At first it may be tempting to suggest that these pressures should be equal to the systolic arterial pressure. In fact, however, the pulse pressure is virtually entirely damped in the microcirculation so that nothing corresponding to systolic pressure ever appears on the venous side of the circulation. The venous pressure in the outflow-occluded stump is, in fact, equal to mean arterial pressure. This fact, which we have confirmed by direct measurement, also has a sound theoretical basis. The average driving force for flow in the microcirculation is the root-mean-square (RMS) pressure. To the extent that the RMS pressure can be equated to the more traditionally defined "mean arterial pressure", we see that the magnitude of the latter is most closely related to measurable dynamic events in the limb, particularly the rate of change of volume as a result of blood flow. Thus, mean arterial pressure, which has heretofore been defined by acclamation or measured by an electronic integration, in the so-called BFS theory emerges as a real physical force. Our current concept of the reality of MAP is fundamental to understanding the physiologic mechanisms of the oscillometric and statometric methods of measuring blood pressure.

### 3.4 Oscillometric

In the oscillometric blood pressure method, it has been empirically observed that as cuff pressure is decreased from an initial pressure above systolic pressure, the lowest cuff pressure which gives maximum oscillations in the cuff pressure is MAP[11]. Systolic and diastolic pressures are estimated using an empirically-based dual slope estimation technique. The MAP determination, furthermore, has never been physically or physiologically explained. We believe that the BFS Theory describes the maximum oscillations occurring at the point where cuff pressure is equal to MAP.

The oscillations are small and unchanging when the cuff is above systolic pressure. This is caused by the "leading edge" effect. The edge of the cuff proximal to the occlusion is influenced by pulses in the proximal artery, or "stump". As the cuff deflates through systolic pressure, the pulse amplitudes increase. This is caused by the pulses partially passing through the occlusion and creating a pulsating pressure in the cuff. The venous pressure distal to the cuff increases, and approaches the mean arterial pressure distal to the occlusion. When the cuff pressure is equal to MAP, the arterial pressure and venous pressure distal to the occlusion are also equal to MAP. As the cuff deflates a very small amount, the venous pressure (MAP) will be greater than the cuff pressure (just below MAP). Outflow from the distal limb commences. Since the cuff is sensing pulses which represent a net flow

phenomenon, the pulses sensed by the cuff start to decrease in amplitude. Thus, the pulse amplitudes measured by the cuff have a maximum at MAP.

### 3.5 Statometric

The statometric technique is a blood pressure measurement technique that uses a low frequency (on the order of 0.15 Hz) limb volume change to determine blood pressure[12]. In this technique a limb volume sensor is placed distal to an occlusion cuff. As the occlusion cuff is inflated above systolic pressure the distal limb becomes ischemic. Within a few seconds the normal autoregulatory mechanisms attempt to compensate for the ischemia. In effect the vascular capacity is increased by vasodilation. As the occlusion cuff is slowly deflated through systolic pressure, the distal limb volume starts to increase. As the cuff pressure is further reduced to the level of MAP there is a rapid change in the rate of filling of the distal limb. This occurs because, as predicted by the BFS theory, venous outflow will begin when cuff pressure is slightly less than MAP. At this point the limb volume vs. pressure curve shows an inflection point which corresponds to the beginning of outflow and an occlusion cuff pressure equal to MAP. Thus systolic pressure is the point at which the limb volume first begins to increase and MAP is the point at which the rate of increase changes most rapidly. Diastolic pressure is calculated using the relation, $D=\frac{1}{2}(3M-S)$.

# 4 DESIGN

Fig. 1 shows the schematic diagram of the fluidic system of the test fixture. This system is separated into two sections. The first section is called the heart (inside the wood box), and the second is called the arm (limb).

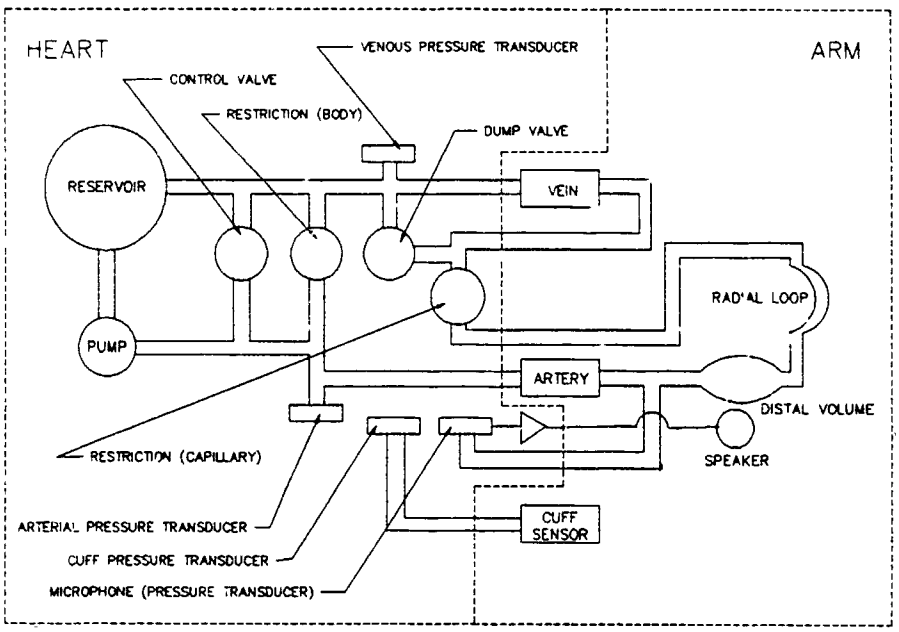



**Figure 1.** Schematic Diagram of the Artificial Arm and Heart

## 4.1 Heart

The functional diagram of the heart section is shown in Fig. 2. The inlet of a gear pump[b] is attached to a reservoir (simulating the venous pool). The level of the fluid in the reservoir creates a pressure head on the inlet of the pump representing central venous pressure. The outlet of the pump is attached to the arm as well as a restriction simulating the peripheral resistance of the rest of the body.

The reservoir is heated with an electrical heating element. This element is controlled with a bang-bang controller. The temperature[c] measured in the reservoir is compared to a

---

[b] Cole Parmer: Pump Head Model #7002-18, Motor Model #7144-91, and Power Supply Model #2630-90.

[c] The temperature sensor is Analog Devices AD590.

threshold temperature of 37°C. The comparator output controls the heater. To prevent overheating of the reservoir, the fluid is constantly circulated at a minimal rate. This prevents any potential for localized temperature gradients and possible damage to the reservoir or heater. When the heater is on, the red light on the front of the wood box is on. This is to indicate to the user that the heater system is active.

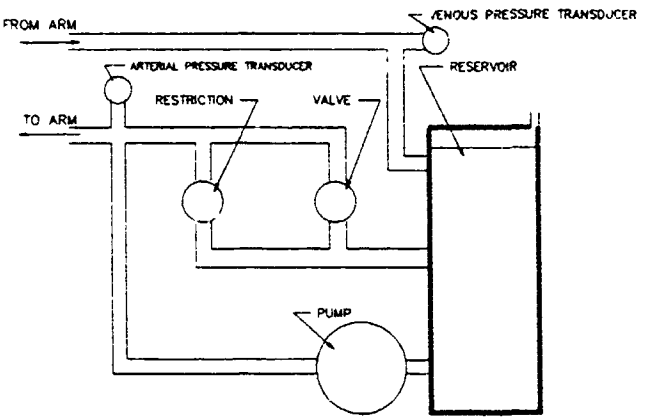



**Figure 2.** Functional Diagram of the Heart

A control system is required for maintaining the arterial pressure stable and equal to the desired pressure. The controller consists of three parts. The first part is an open loop controller The computer controls the pump via a digital-to-analog converter (DAC); the desired pressure is converted to voltage and applied to the pump power supply. The conversion was determined by applying a voltage to the pump and measuring the arterial pressure[d]. The empirically determined equation for this is:

$$V_{pump} = -0.1334[V_{press}^{3}] + 1.2178[V_{press}^{2}] - 1.2401[V_{press}] + 1.2732.$$

The desired arterial pressure waveform is calculated using the following equation.

$$P(t) = \Sigma_i[A_i\cos(t \cdot HR/2\pi) + B_i\sin(t \cdot HR/2\pi)]$$

where $i = \{1,2,3,4,5,6\}$

$A_i = \{-10.14,-0.30,-0.15,1.08,0.21,0.42\}$

$B_i = \{6.79,-3.29,-0.97,-0.24,0.07,0.63\}$

The second part of the control system is a bang-bang controller. A solenoid valve[e] is controlled by comparing the real arterial pressure and the desired arterial pressure. The

---

d The pressure transducers are Data Instruments Model #EA9300002, 0-15 psig.

e Asco Model #8262C6 12V normally closed valve.

inherent capacitance in fluid inertia, as well as the capacitance of the simulated artery, result in a time delay from the pump applying pressure to the pressure sensed by the arterial transducer; this time delay was measured to be 120 milliseconds. Therefore, the desired pressure signal must be temporally synchronized to the real pressure. The computer delays the desired signal by 120 msec and then sends, via a second DAC, the desired pressure to the bang-bang comparator. Fig. 3 shows the functional diagram of this concept.

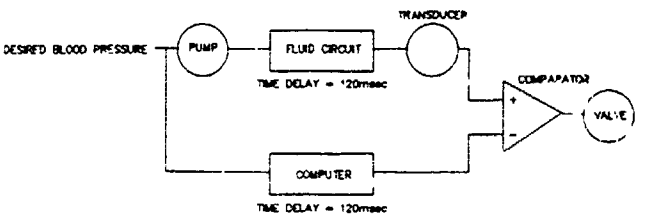



**Figure 3.** Fluid Delay Compensation Circuit

The third part of the control loop is an adaptive controller implemented in the software. Since the arm and the "rest of the body" are parallel (and very different) circuits, the actual arterial pressure varies as the arm is partially occluded (taken out of the circuit). The error between the desired pressure (systolic and diastolic) and the actual pressure (systolic and diastolic) is calculated every two pulses. This error is used to modify two coefficients which are applied to the oscillating component and the offset component of the desired waveform applied to the pump. This adaptively controls the pump depending on the actual and desired arterial pressures. These three parts of the control system are necessary to control this complex and dynamic circuit.

In order to calibrate the device, application of the calibration assembly allows pressurization of the reservoir and the cuff. This static pressure is transmitted to the whole fluidic system as well as the cuff pressure transducer. The arterial, venous, and cuff pressure transducers can be calibrated simultaneously.

## 4.2 Arm

In order to provide an anthropomorphic test fixture, the arm is constructed from a modified IV training arm. Fig. 4 shows the fluid schematic. The plumbing consists of a simulated artery, an expandable bag (simulating the distal limb volume), a restriction (simulating a capillary), and a simulated vein. Fig. 5 shows the side view of the limb. The artery

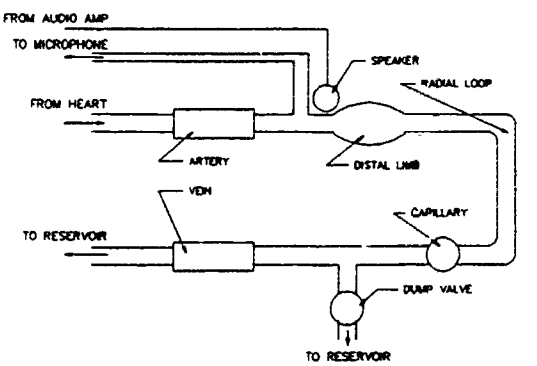



**Figure 4.** Functional Diagram of the Arm

and vein are located on the upper arm. A chamber of oil, covered by a rubber membrane, is the cuff pressure sensor. This is fluidically coupled to a pressure transducer. The cuff pressure is transmitted through the skin and membrane to the oil, and in turn, the transducer. Another pressure transducer is used as a microphone. It is attached to the arterial line distal to the artery. Sounds generated by the artery (Korotkoff sounds) are sensed by this transducer, amplified, and played back through a speaker located under the skin at the elbow on the arm.

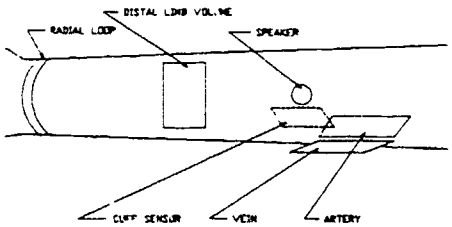



**Figure 5.** Physical Location of Arm Components

There are two major technological advances in the limb - the occluding vessel and the simulation of ischemia.

4.2.1 Vessels

The initial design of the occluding vessels (artery and vein) was a flexible tube (penrose rubber). We placed this rubber tube inside a closed chamber of water which had one pressurization port. The rubber tube had water flowing through it at a certain internal pressure. Unlike Kresch & Noordergraaf[13,14], Conrad[15], and Holt[16], who imply that flow is zero when internal pressure is equal to external pressure, we observed that the flow was not zero when the chamber pressure was equal to or greater than the internal tube pressure (occlusion). We discovered that (as did Kresch and Noordergraaf, 1972) as the rubber tube collapsed, extremely small channels remained at the outer ends (Fig. 6). The chamber pressure had to be very large in order to collapse the small channels.

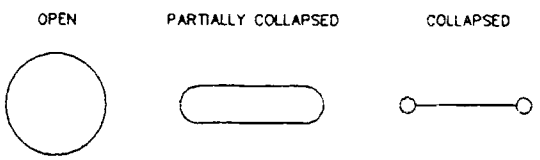



**Figure 6.** Illustration of Tube Collapse

Since penrose rubber was not suitable as an *in situ* occluding vessel, we developed our own occluding vessel. Fig. 7 shows the mechanical drawing of this vessel.

The fluid pressure (e.g., arterial pressure) causes the thin rubber membrane to "bulge" and fluid is allowed to pass through the vessel. If the occluding pressure (from the cuff) is equal to the fluid pressure, the fluid cannot create the bulge, and the vessel is occluded. The forces arising from the fluid pressure and the external pressure are in balance (since the areas on which the forces act are equal). As one would expect, the arterial pulses can be measured in the oscillating movement of the rubber membrane, and in turn, in an occluding cuff applied to this vessel. This design results in a practically-realizable Starling resistor[17].

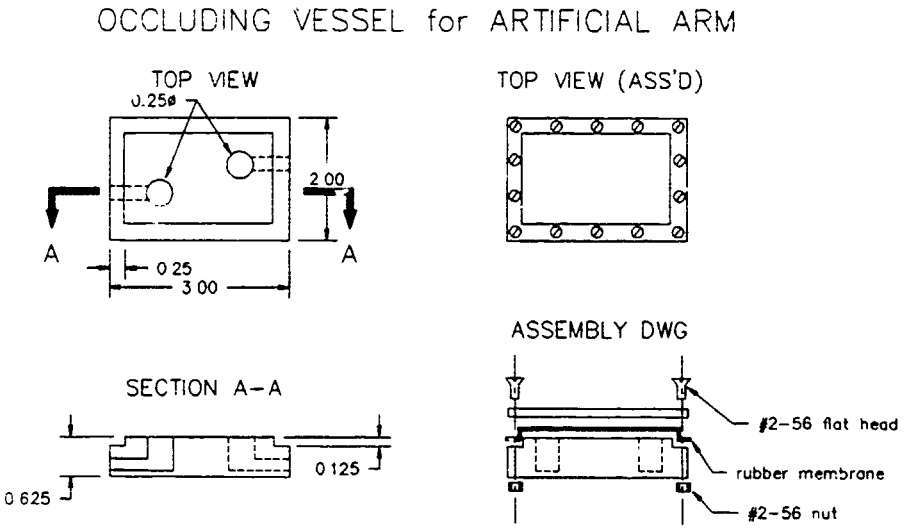



**Figure 7. Mechanical Drawing of GMS Starling Resistor**

### 4.2.2 Ischemia

In the human, the arm becomes ischemic when the artery is occluded and the distal portion of the arm ceases to receive blood. This causes vasodilatation, which decreases the intravascular pressure. Blood flows into the arm when the cuff pressure is deflated below systolic pressure, because of this intravascular pressure decrease. If the intravascular pressure, distal to the occlusion, remained constant (at MAP), then the inflow of blood (when the occlusion is no longer total) would be prevented. This is because, in order to create significant flow, there must be a pressure drop. In the prototype test fixture, ischemia is simulated using a valve[f] (dump valve in Fig. 1). This valve, when opened, allows some of the fluid in the limb distal volume (see Fig. 1) to empty directly into the reservoir. This

[f] Asco Model #82628210 12V normally closed valve.

volume reduction causes the "intravascular" pressure to decrease. In effect, the same result as vasodilatation has been created. The simulation of ischemia is essential to the operation of the three blood pressure measurement techniques. Each of these techniques operates on the fact that as the cuff deflates below systolic pressure, the blood flow into the limb becomes significant.

### 4.3 Electrical Hardware

The electrical schematics are located in Appendix A. The first schematic is the computer interface board. There are anti-aliasing filters for each transducer. There is an eighth order low pass Butterworth filter for the arterial transducer. The arterial transducer signal requires the use of a Butterworth filter to reject the oscillations due to the control valve. There are fourth order low pass Bessel filters for the other transducers. Also on the board are two comparators (essentially bang-bang controllers); the valve controller and the heater controller. The second schematic shows the circuit located in the arm box, consisting of the valve drivers, the transducer drivers, and the audio amplifier for the Korotkoff sounds. There are fuses to protect the electronics from any current surge or electrical short. They can be found on the power supply box on the computer stand. The fuse next to the power cord (on the AC side of the power transformer) is a four amp fuse, and the other fuse (on the DC side of the power transformer and located next to the round connector) is an eight amp fuse.

### 4.4 Computer Software

The software code is located in Appendix B. The program is written in C, and it includes Halo graphics software and C-Tools (used for the interrupt service routine support). The program is modular to allow for modification if necessary. Its function can be seen in the video manual showing the setup, calibration, operation, and troubleshooting of the test fixture.

The program consists of three parts. The first part is the vital signs menu and help functions. This utilizes the Halo graphics routines and the arrow and enter keys on the keyboard to choose vital signs parameters or obtain on-line help. The second part is the main shell of the program. This updates the screen approximately every second. The screen consists of two traces, a process diagram, a manometer, a user-entered header, the time and date, and the numerical information describing the arm. Both the on-line help and the video tape are necessary to view for further understanding of the operation of the arm. The last part is the interrupt service routine. This acquires data and controls the pump and control valve. This routine is entered as an interrupt to the processor every twenty milliseconds.

# 5 RESULTS

The prototype test fixture simulates the circulatory physiology of a human limb. The auscultatory technique, the oscillometric technique, and the statometric technique measure the arterial pressure in the arm (controlled by the pump and fluidic system). This prototype test fixture eliminates all of the inherent problems of testing noninvasive vital signs monitors with many human subjects.

With the use of this test fixture, one does not need many human subjects and multiple measurements. One can set various blood pressures and heart rates, place the fixture in any environment, and test the vital signs monitor accurately.

The problem of the vital signs being very labile is no longer an issue here. The test fixture maintains the desired blood pressure. There is no physical or psychological stress. The fixture has been tested for eight hours continuously, and the blood pressure remained constant (within 2 mmHg) during that time.

The test fixture does not need time to equilibrate after a blood pressure measurement. The user can take vital signs measurements several seconds apart (after the previous one ends).

The problem of cuff placement is magnified in many subjects. The cuff must be applied multiple times (as many as there are subjects). The cuff needs only to be applied once on the test fixture.

The test fixture makes no voluntary movements. Any movements are passive and caused by the user or the environment (shake table or vehicle). The patient movement problem does not appear.

There exists no pressure error due to differences in vertical height of the heart and the measuring site. The arterial pressure transducer is on the same vertical level as the artery. The pressure error is zero.

Due to time and budgetary constraints, this prototype test fixture was minimally tested. There are some problem areas with the arm that the user must understand to properly use it. The next section discusses these problems in detail.

# 6 PROBLEM AREAS AND SUGGESTIONS FOR FURTHER WORK

There are several problem areas that we have encountered in this development effort. Time and budgetary constraints prohibited us from investigating these areas more completely. In order for one to use the arm properly, one must fully appreciate these problems areas and their effects on the function of the arm.

The first problem area is the control loop. The adaptive part of the control loop depends solely on the heart rate, since the systolic and diastolic pressures are computed every two pulses. Overshoots are increased and small oscillations are extended in time as the heart rate decreases. We have observed that in the prototype test fixture, the oscillometric technique operates more inconsistently as the heart rate is lowered. We believe that further development will allow fine tuning and more complete testing of the control loop, and in turn, allow consistent operation of the oscillometric technique over the whole heart rate range.

The second problem area is the cuff pressure transducer's nonlinearity and cuff placement criticality. This transducer consists of a chamber of oil covered with a thin rubber membrane. This chamber is fluidically coupled to a pressure transducer. The cuff pressure couples to this chamber through the skin. We have found that the overall (cuff to pressure transducer) cuff pressure measuring system is slightly nonlinear. The cuff pressure is higher than the transducer measurement for low cuff pressures. We believe that further development will allow us to redesign the cuff pressure transducer chamber so that the system is linear, or as a minimum, deterministic. The user must calibrate the cuff transducer immediately after each and every cuff application. We have found that the cuff placement is extremely critical to the pressure transmission from the cuff to the transducer. A redesign of this chamber will eliminate the need for the frequent calibration.

The third problem area is the noise of the solenoid valve opening and closing. This sound propagates through the fluid lines and is sensed by the microphone. This sound, along with the Korotkoff sounds, are amplified. The valve sounds result in the difficulty in hearing the Korotkoff sounds. We believe that further development will permit us to obtain a specially fabricated valve with rubber stops (instead of the metal ones); this will eliminate the valve sounds.

The fourth problem area is the "dump valve" control. This valve is modelling the ischemia, and is controlled by the relationship between the cuff pressure and the systolic pressure. When the cuff pressure is greater than systolic pressure, there is no flow in the arm. A flow transducer (or differential pressure measurements) in the arterial line would allow better control of this valve. We have found that the problem of the cuff pressure transducer magnifies the problem here. If the cuff pressure transducer is inaccurately measuring the cuff pressure, the "dump valve" is opened at the wrong time, and the distal limb is emptied too early or too late. Both of these scenarios lead to the inconsistent operation of the oscillometric and the statometric techniques. Further development might result in another method (fluidic) of modelling the ischemia.

The prototype test fixture is fragile and minimally tested. The scientific progress in understanding and modelling the underlying physical and physiological principles of noninvasive blood pressure measurement is considerable. This, unfortunately, minimized the time for ruggedizing and testing the final test fixture.

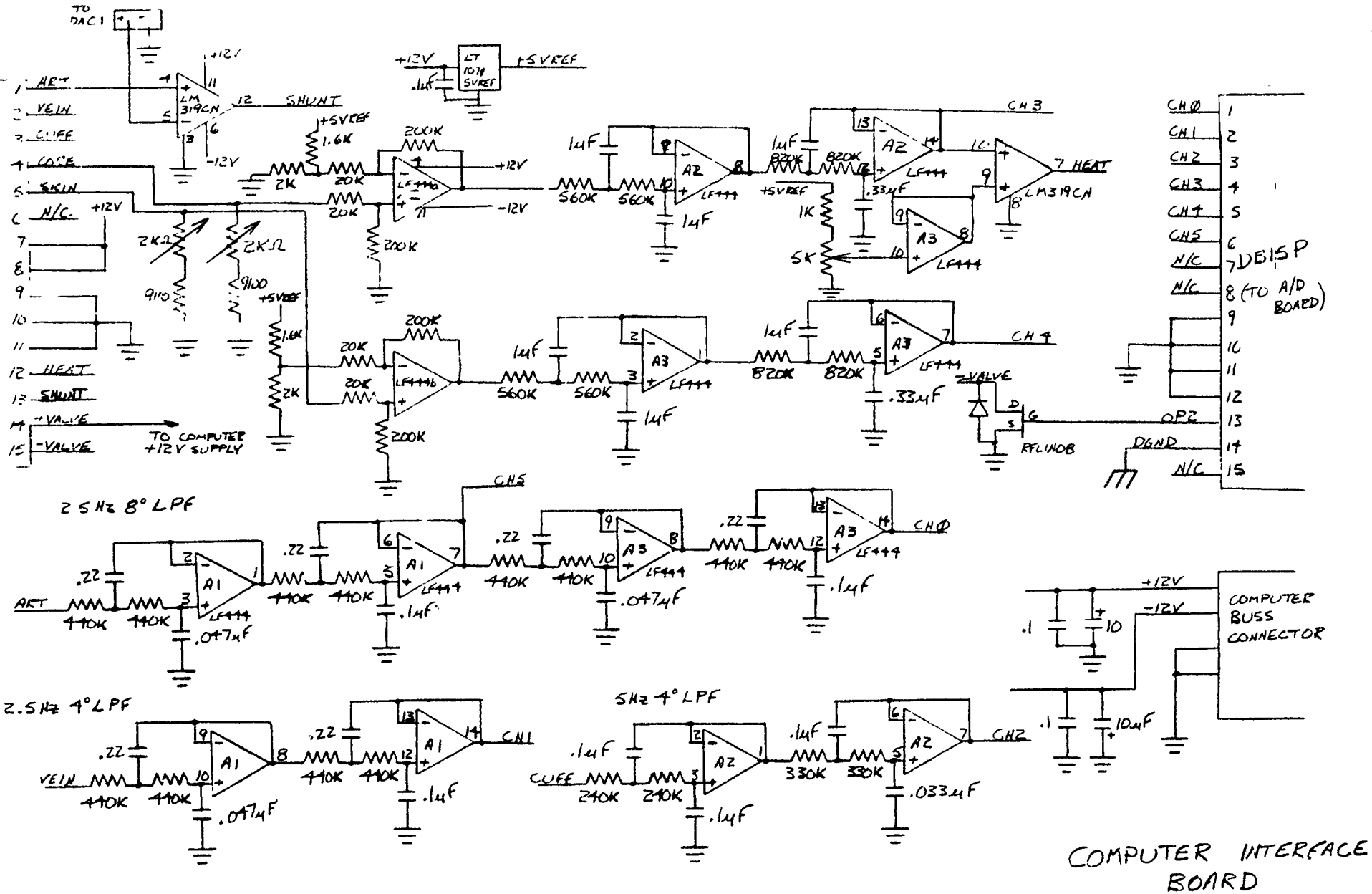
# 7 CONCLUSIONS

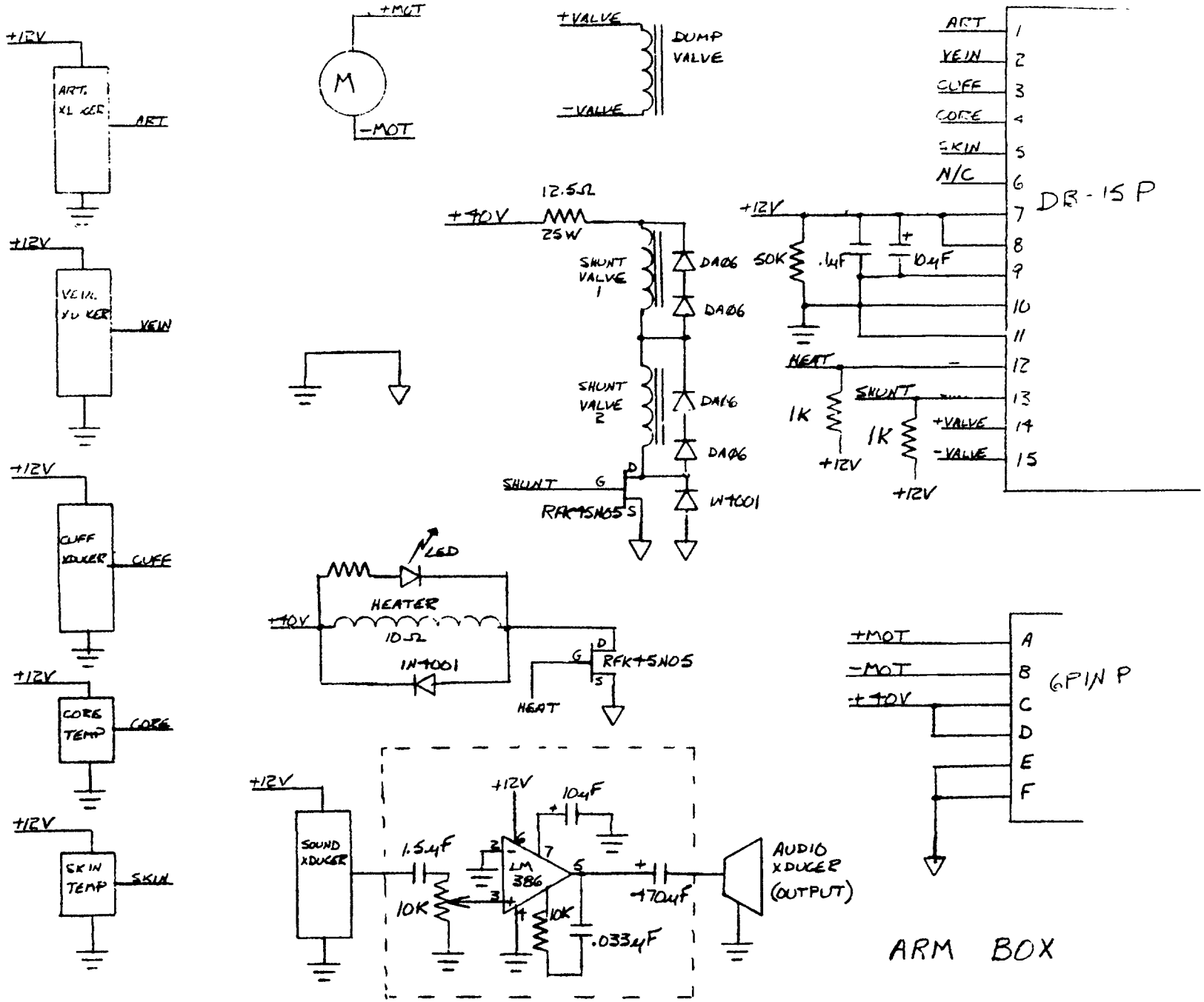
The prototype test fixture fills a void in checking, calibrating, and maintaining vital signs monitors. This test fixture will allow accurate monitoring and control of the simulated dynamic arterial pressure; this eliminates all of the associated problems in testing noninvasive vital signs monitors (see the Results section). The test fixture has a great potential in both the military and civilian environments; it should allow routine checking of all vital signs monitors, and it should improve maintenance of vital signs monitors.

# 8 REFERENCES


1. Finnie, K.J.C., Watts, D.G., and Armstrong, P.W., Biases in the measurement of arterial pressure, Critical Care Medicine, 12:965-968, 1984.
2. Gould, B.A., Hornung, R.S., Altman, D.G., Cashman, P.M.M., and Raftery, E.B., Indirect measurement of blood pressure during exercise testing can be misleading, Br. Heart J., 53:611-615, 1985.
3. Manning, D.M., Kuchirka, C., and Kaminski, J., Miscuffing: inappropriate blood pressure cuff application, Circulation, 68:763-766, 1983.
4. Smith, N.T., Wesseling, K.H., and de Wit, B., Evaluation of two prototype devices producing noninvasive, pulsatile, calibrated blood pressure measurement from a finger, J. Clin. Monit., 1:17-29, 1985.
5. Maurer, A.H., Korotkoff sound filtering for automated three-phase measurement of blood pressure, American Heart J., 91:584-591, 1976.
6. Korotkoff, N.S., On the subject of methods of measuring blood pressure, Bull. Imp. Milit. Med. Acad. St. Petersburg, 11:365, 1905.
7. Gittings, J.C., Auscultatory blood-pressure determinations, Arch. Intern. Med., 6:196, 1910.
8. Aliker, M., and Raman, K.R., Korotkoff sounds at diastolic pressure - a phenomenon of dynamic instability of fluid filled shells, Int. J. Solids Structures, 3:467, 1966.
9. Conrad, W.A., McQueen, D.M., and Yellin, E.L., Steady pressure flow relations in compressed arteries: possible origin of Korotkoff sounds, Med. & Biol. Eng. & Comput., 18:419-426, 1980.
10. Dittrich, H.C. and Slutsky, R.A., Radionuclide analysis of the forearm venous pressure-volume relationship: response to nitroglycerin, Am. Heart J., 107:733-737, 1984.
11. Ramsey, M., Noninvasive automatic determination of mean arterial pressure, Med. & Biol. Eng. & Comput., 17:11-18, 1979.
12. Samaras, G.M., Blaumanis, O.R., and Van Horn, H.W., Noise-immune blood pressure measurement technique and system, U.S. Patent No. 4,649,928, 1987. Other patents pending.
13. Kresch, E. and Noordergraaf, A., A mathematical model for the pressure-flow relationship in a segment of vein, IEEE Trans. Biomedical Eng., BME-16:296-307, 1969.
14. Kresch, E. and Noordergraaf, A., Cross-sectional shape of collapsible tubes, Biophysical J., 12:274-294, 1972
15. Conrad, W.A., Pressure-flow relationships in collapsible tubes, IEEE Trans. Biomedical Eng., BME-16:284-295, 1969.
16. Holt, J.P., Flow through collapsible tubes and through "in situ" veins, IEEE Trans Biomedical Eng., BME-16:274-283, 1969.
17. Knowlton, F.P. and Starling, E.H., The influence of variations of temperature and blood pressure on the performance of the isolated mammalian heart, J. Physiol., 44:206-219, 1912.

# APPENDIX A - Electrical Schematics


TO DAC 1
ART
VEIN
CUFF
SKIN
N/C.
HEAT
SHUNT
+VALVE
-VALVE
TO COMPUTER +12V SUPPLY
LM 319CN
SHUNT
+5VREF
LT 1019 5VREF
1.6K
2K
20K
200K
2K Ω
9.1K Ω
9.1K
560K
1μF
820K
1K
5K
.33μF
.1μF
.047μF
.22
440K
330K
240K
.033μF
10μF
10
.1
LF444
A1
A2
A3
CH3
CH4
CH5
CH0
CH1
CH2
HEAT
-VALVE
RFLINOB
OPZ
DGND
+12V
-12V
7 DB15P
8 (TO A/D BOARD)
COMPUTER BUSS CONNECTOR
2.5Hz 8° LPF
2.5Hz 4° LPF
5Hz 4° LPF
COMPUTER INTERFACE BOARD


+12V
ART. XL XDCR
ART
+12V
VEIN. XD XDCR
VEIN
+12V
CUFF XDUCER
CUFF
+12V
CORE TEMP
CORE
+12V
SKIN TEMP
SKIN
+MOT
M
-MOT
+VALVE
DUMP VALVE
-VALVE
+10V
12.5Ω
25W
SHUNT VALVE 1
SHUNT VALVE 2
DA06
DA06
DA06
DA06
IN4001
SHUNT
G
D
S
RFK45N05
LED
HEATER
10Ω
IN4001
HEAT
RFK45N05
ART 1
VEIN 2
CUFF 3
CORE 4
SKIN 5
N/C 6
DB-15P
+12V 7
8
9
10
11
HEAT 12
SHUNT 13
+VALVE 14
-VALVE 15
+12V
50K
.1μF
10μF
1K
1K
+12V
+12V
+MOT A
-MOT B
+10V C
D
E
F
6PIN P
+12V
SOUND XDUCER
1.5μF
10K
+12V
10μF
LM 386
10K
.033μF
470μF
AUDIO XDUCER (OUTPUT)
ARM BOX

# APPENDIX B - Software Code

```
/*      ARM             DEFINITIONS     */

#define FRAME           0x01    /*  The IRR bit set by frame clock  */
#define A2D             0x02    /*  The IRR bit set by A/D EOC      */
#define DELAY           0x04    /*  The IRR bit set by write delay */
#define ART_P           0x00    /*      Arterial pressure      */
#define CUFF_P          0x01    /*      Cuff pressure       */
#define VEIN_P          0x02    /*      Venous pressure     */
#define CORE_T          0x03    /*      Core temperature    */
#define SKIN_T          0x04    /*      Skin temperature    */
#define MAX_MEASURED    500
#define DAS8_BASE_ADDR  0x310   /*  Base address of DAS-8 board  */
#define A2D_LO_BYTE     0x310   /*  A/D converter data low byte  */
#define A2D_HI_BYTE     0x311   /*  A/D converter data hi byte   */
#define START_8_BIT     0x310   /*  Start 8-bit conversion addr  */
#define START_12_BIT    0x311   /*  Start 12-bit conversion addr */
#define DAS8_STATUS     0x312   /*  DAS-8 status register               */
#define DAS8_CONTROL    0x312   /*  DAS-8 control register       */
#define CNT_0           0x314   /*  8254 counter #0 address      */
#define CNT_1           0x315   /*  8254 counter #1 address      */
#define CNT_2           0x316   /*  8254 counter #2 address      */
#define CNT_CTRL        0x317   /*  8254 control reg address     */
#define INT_CHAN        0x05    /*  Hardware interrupt # used    */
#define EOI_ADDR        0x20    /*  8259 control register               */
#define PIC             0x20    /*  8259 control register               */
#define ENABLE_INTS     0x08    /*  "Enable interrupts" command  */
#define D1_OUT_HI       0x10    /*  "Set D1 out hi" command      */
#define D2_OUT_HI       0x20    /*  "Set D2 out hi" command      */
#define D3_OUT_HI       0x40    /*  "Set D3 out hi" command      */
#define D4_OUT_HI       0x80    /*  "Set D4 out hi" command      */
#define     DAC0_LO     0x308   /*  DAC0's lo byte address       */
#define     DAC0_HI     0x309   /*  DAC0's hi byte address       */
#define     DAC1_LO     0x30A   /*  DAC1's lo byte address       */
#define     DAC1_HI     0x30B   /*  DAC1's hi byte address       */
#define     DAC_MAX     0xFFF   /*  Maximum DAC output           */
#define     ECG_LO      DAC0_LO /*  DAC0 is the ECG output       */
#define     ECG_HI      DAC0_HI /*  DAC0 is the ECG output       */
#define     HEATER      DAC1_LO /*  DAC1 is the Heater output    */
#define FRAME_ERR       0x01    /*  Frame overlap  */
#define A2D_ERR         0x02    /*  A/D overlap    */
#define DELAY_ERR       0x04    /*  Delay overlap  */
#define CLOSE_ERR       0x08    /*  Process overlap */
#define INDET_ERR       0x10    /*  IRR = 0          */
#define UNEXP_ERR       0x20    /*  Unexpected A/D int.   */
```

```
/*

                TEST FIXTURE FOR VITAL SIGNS MONITORS

*/

#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <halo.h>
#include <time.h>
#include <malloc.h>
#include <stdio.h>
#include <butil.h>
#include <bkeybrd.h>
#include <bscreens.h>
#include <bstrings.h>
#include <bintrupt.h>
#include <das8_defs.h>
#include <err_defs.h>              /* Err code defs */
#include <dac_defs.h>              /* D/A addrs & functs */
#include <datadefs.h>              /* Conversion constants */

#define PI        3.1415926535
#define GRAPHS_EVERY_X   2
#define UPDATE_X_LOOPS   2     /* Max value for this is 10 due to
                                                    hr_, sys_ , dias_, map_
                                                    arrays declared with lengths of
                                                    10. */
#define MAX_LENGTH 300             /* Maximum length of the bp and ecg waveform
                                                    arrays.  Change this value from 120 if
                                                    the sample interval changes from 50 msec
                                                    or the minimum heart rate changes from
                                                    30bpm. */

#define N_A2D_CHAN       5
#define INT_FREQ        50.0                        /*  Int freq  */
#define BLEED_TIME       75
#define STACKSIZE        0x2000
#define NSTKS            1
#define INT_TYPE INT_CHAN + 8              /*  Int vector type */

#define SPEC_EOI    0x60 | ( 0x07 & INT_CHAN )
#define YES  1
#define NO   0
#define ON   0
#define OFF  1



/* Declare subroutines */
void main(void);

void set_up_vs_settings_menu(int);
void set_up_vs_query(void);
void obtain_vs_settings(void);
void set_up_manometer_disp(void);
void set_up_icu_disp(void);
void draw_graph_labels(int, int, int);
void set_up_process_disp(void);
void draw_loop_outline(void);
```

```
void set_up_options_bar(void);
void help_system(void);

void update_manometer(float);
void update_icu(int,int,int,int);
void update_process(int, int, int, int, int);
void draw_arm_outline(void);
void update_header(char *);
void update_options(int);
void gms_logo(void);
int read_value(char *, int, int, float *);

int matherr(struct exception *);

/* Declare interrupt subroutines. */

int enable_ints( int );                          /* Enables ints on DAS-8 board. */
int disable_ints( int );                         /* Disables ints on DAS-8 board.*/
int set_imr( int, int );                         /* Programs the 8259 mask reg  */

int set_timer( int, int, double, double );       /* Inits 8254 counter */
int read_status( int );                  /* Reads the status word of 8254   */


/* Declare and initiate values */

int sys_valve;
int sys_check;
int dias_valve;
double press_map;
int minalpha;

int main_restart = 0;
int manom_restart = 0;
int icu_restart = 0;
int process_restart = 0;
int drw_grph_restart = 0;
int isr_restart = 0;
int calflag = 0;
int exit_flag = 0;

int mode = 4;
static char filename[] = {"HALOEVME.DEV"};
float min_x = (float) 0.0;
float min_y = (float) 0.0;
float max_x = (float) 1000.0;
float max_y = (float) 1000.0;

int blue = 0, blue_val = 1;
int white = 1, white_val = 63;
int purple = 2, purple_val = 21;
int red = 3, red_val = 4;
int light_gray = 4, light_gray_val = 7;
int navy = 5, navy_val = 8;
int dark_gray = 6, dark_gray_val = 56;
int black = 7, black_val = 0;
int light_blue = 8, light_blue_val = 59;
int yellow = 9, yellow_val = 54;
int bright_pink = 10, bright_pink_val = 61;
int bright_blue = 11, bright_blue_val = 43;
int bright_orange = 12, bright_orange_val = 38;
```

```
int bright_green = 13, bright_green_val = 26;
int gms_blue = 14, gms_blue_val = 9;
int skin_pink = 15, skin_pink_val = 39;

float asp, aspect=(float)1.7;
int hor_pix, ver_pix;

int degrees = 1;

int result;

static char fontname[] = {"HALO104.FNT"};
static char fontname2[] = {"HALO106.FNT"};
int page1 = 1;
int page2 = 2;

int new_header = 1;

int data[MAX_MEASURED][10];
int data_base[MAX_MEASURED];

int most_recent_data;
int most_recent_plotted;
int ar = 0, ec = 5, cu = 1, ve = 2;
int pp = 6, sk = 4, co = 3, gr = 7;
int di = 8, ra = 9;
int max_points = MAX_MEASURED - 1;
int update;
int frozen;

int period_msec, num_pts_in_period;
int sample_interval = (int) (1000/INT_FREQ);
int num_pcs, slope;
int current_length;
double dtemp0;
double dtemp1;
int t_off = 120;
int alpha = 100;
int beta = 100;
int ecg_des[MAX_LENGTH];
int bp_des[MAX_LENGTH];
int bp_d0[MAX_LENGTH];
int bp_d1[MAX_LENGTH];

int hr_set;
int sys_set;
int dias_set;
double press_set[3];
double art_cal;
double cuf_cal;
double ven_cal;

char header[42] = "GMS ENGINEERING CORPORATION VSM TEST";

int last_measured_x;
int last_averaged;

/*      Definitions of the data conversion constants            */

int     k_art_p  = 512;          /*  arterial pressure (mmHg/V)  */
int     k_cuff_p = 531;  /*  Cuff pressure     (mmHg/V)  */
```

```
int     k_vein_p = 483;    /*  Venous pressure   (mmHg/V)  */
int     k_core_t = 24;    /*  Core temperature  (degC/V)  */
int     k_skin_t = 24;    /*  Skin temperature  (degC/V)  */
int     k_ecg    = 1;     /*  Electrocardiogram (  mV/V)  */
int     off_art_p =2007;          /*  arterial pressure (mmHg/V)  */
int     off_cuff_p =2706;  /*  Cuff pressure     (mmHg/V)  */
int     off_vein_p =3487;  /*  Venous pressure   (mmHg/V)  */
int     off_core_t       = 76;    /*  Core temperature  (degC/V)  */
int     off_skin_t       = 34;    /*  Skin temperature  (degC/V)  */
int     off_ecg          = 0;     /*  Electrocardiogram   (mV/V)  */


int over = 2;
int delta_over = 2;
int max_over = 20, min_over = 2;
int at_low_limit = 1;
int at_up_limit = 0;

int low_pitch = 150;
int mid_pitch = 450;
int high_pitch = 700;
int short_duration = 2;
int mid_duration = 5;
int long_duration = 10;

/* Variables for the interrupt portion of the program. */

int frame;                                  /*      The data frame #  */
int end_frame;                          /* "End of frame" flag   */
int dac_v;                                  /* DAC voltage fc. motor */
int last_measured;
int art_press;                          /*  The most recent art_p[]          */
int a_p_index;                          /*   Points to the next art_p loc    */
int cuf_press;                          /*  The most recent cuff_p[]          */
int c_p_index;                          /*  The next cuff_p datum to be entd */
int vein_press;
int v_p_index;
int core_temp;
int c_t_index;
int skin_temp;
int s_t_index;
int bp_ind;

int major_error, basic_error;
int num_points_msg_on;
int message_on;
int num_errors;
int error_num;
char *pmessage;
static char lo_arter_message[] = { " Arter press > 330 mmHg." };
static char lo_cuff_message[] = {  " Cuff press > 330 mmHg. " };
static char hi_arter_message[] = { " Arter press > 360 mmHg." };
static char hi_cuff_message[] = {  " Cuff press > 360 mmHg. " };
static char core_message[] = {  "   Core temp > 43'C.    " };
static char skin_message[] = {  "   Skin temp > 43'C.    " };
static char pps_message[] = { " Motor too fast too long." };
```

```
void arm_isr( ALLREG *, ISRCTRL *, ISRMSG* );     /*  The isr.   */
char    *isr_id = "ARM ISR 9/26/88 ";     /*  ISR signature.   */
ISRCTRL isr_ctrl;                                 /*  Control block for the isr.        */

int motor_v;                              /*  The motor velocity in rpm         */

int das8_state = 0;                               /*  Contents of the DAS-8 control register*/

static ISRCTRL far *pisr_ctrl;                    /* pointer sent to isinstal() */
static char isr_stack[ STACKSIZE*NSTKS ];  /* ISR stack:must be an auto var*/

void main()
{
FILE *stream;

        int pps, cuff, venous, core, skin, arter;
        int inc_dec;
        int curs_pos, ch, newkey;
        int loop_counter;
        int loop_cntr;
        int i,j,k;
        int graph;
        static int freeze, lock;

        int check_settings;
        static char answer[] = "Y";

        double flt_num_pts, single, doubleit, triple;
        int single_leftover, doubleit_leftover, triple_leftover;
        double sin_interval, angle_radians;
        int multiplier;
        double range_bp;

        static double coeff_a[6] = {-10.14, -0.30, -0.15, 1.08, 0.21, 0.42};
        static double coeff_b[6] = {6.79, -3.29, -0.97, -0.24, 0.07, 0.63};
        static double coeff_c[6];
        static double coeff_d[6];
        static int qrs;
        static int ecg_coeff_a[6] = {0,0,0,10,0,0};
        static int ecg_coeff_b[9] = {-10,0,0,0,10,20,0,0,0};
        double minimum;
        double sum, temp1, temp2;
        int add_on;
        double angle_int;
        static char pressure[3];
        static float cal_height = (float) 30.0;
        static float press_height = (float) 60.0;
        static float cal_x = (float) 20.0;
        static float cal1_y = (float) 800.0;
        static float cal2_y = (float) 750.0;
        static float cal3_y = (float) 700.0;
        static float cal4_y = (float) 650.0;
        static float cal5_y = (float) 600.0;
        static float cal6_y = (float) 550.0;
        static float cal7_y = (float) 500.0;
        static float cal8_y = (float) 450.0;
        static float cal9_y = (float) 400.0;
        static float calap_x = (float) 160.0;
        static float calp_y = (float) 250.0;
        static float calcp_x = (float) 460.0;
        static float calvp_x = (float) 760.0;
```

```
static float calb1_x = (float) 100.0;
static float calb1_y = (float) 200.0;
static float calb2_x = (float) 900.0;
static float calb2_y = (float) 350.0;
static char cal1_msg[] = { "Please attach calibration assembly tc the air port, to" };
static char cal2_msg[] = { "the bulb, to a manometer, and apply the cuff around the" };
static char cal3_msg[] = { "arm.  The pressures will be updated continuously" };
static char cal4_msg[] = { "in order to check the calibration.  If you wish to exit," };
static char cal5_msg[] = { "press E.  If you wish to calibrate the transducers," };
static char cal6_msg[] = { "apply zero pressure and press L.  Then apply 250 mmHg pressure" };
static char cal7_msg[] = { "and press H.  The calibration will not be changed until tne H" };
static char cal8_msg[] = { "is pressed." };
static char cal9_msg[] = { "      ARTERIAL          CUFF          VENOUS" };
static int art_pres_ten, cuf_pres_ten, vein_pres_ten;

void far *int_vec = isgetvec (INT_TYPE);   /* old isr vector  */

int path = 0;
int on_off_toggle;

float error_x1 = (float) 700.0, error_y1 = (float) 765.0;
float error_x2 = (float) 975.0, error_y2 = (float) 815.0;
float error_height = (float) 20.0;

float out_x1 = (float) 0.0, out_y1 = (float) 0.0;
float out_x2 = (float) 1000.0, out_y2 = (float) 1000.0;
float in_x1 = (float) 100.0, in_y1 = (float) 300.0;
float in_x2 = (float) 900.0, in_y2 = (float) 700.0;
float big_height = (float) 70.0;
float question_height = (float) 30.0;
float message_height = (float) 40.0;
float term_x = (float) 105.0, term_y = (float) 725.0;
static char terminate[] = { "Simulation Terminated" };
float quest_x = (float) 70.0, quest_y = (float) 150.0;
static char start_over[] = { "Would you like to start a new simulation? (Y)es or (N)o: " };
float messg_x = (float) 115.0, messg_y = (float) 600.0;

outp(DAC0_HI,0x32);
outp(DAC0_LO,0);
outp(DAC1_HI,0xFF);
outp(DAC1_LO,0xF0);

setdev(filename);
initgraphics(&mode);

setcolor( &black );
clr();

/* Set aspect ratio. */
inqasp(&asp);
inqdrange(&hor_pix, &ver_pix);
aspect = (max_y - min_y) / (max_x - min_x);
aspect *= ( ((float)hor_pix + (float)1.0) /
            ((float)ver_pix + (float)1.0) ) * asp;

setworld(&min_x, &min_y, &max_x, &max_y);

/* Set color pallette. */
setxpal(&light_blue, &light_blue_val);
setxpal(&blue, &blue_val);
setxpal(&purple, &purple_val);
```

```
        setxpal(&red, &red_val);
        setxpal(&light_gray, &light_gray_val);
        setxpal(&navy, &navy_val);
        setxpal(&dark_gray, &dark_gray_val);
        setxpal(&black, &black_val);
        setxpal(&white, &white_val);
        setxpal(&yellow, &yellow_val);
        setxpal(&bright_pink, &bright_pink_val);
        setxpal(&bright_blue, &bright_blue_val);
        setxpal(&bright_orange, &bright_orange_val);
        setxpal(&bright_green, &bright_green_val);
        setxpal(&gms_blue, &gms_blue_val);
        setxpal(&skin_pink, &skin_pink_val);

        setdegree( &degrees );

        setfont(fontname);

        setscreen(&page1);
        gms_logo();
        display(&page1);


/*                      START from previous run  */

start:
        /*  Initialize variables */

        if (main_restart) {
                manom_restart = 1;
                icu_restart = 1;
                process_restart = 1;
                drw_grph_restart = 1;
                isr_restart = 1;
                main_restart = 0;
        }
        on_off_toggle = ON;
        num_points_msg_on = 0;
        message_on = 0;
        calflag = 0;
        major_error = 0;
        basic_error = 0;
        num_errors = 0;
        sys_valve = 999;
        sys_check = 0;
        dias_valve = 0;
        press_map = 0.0;
        frozen = 0;
        num_pts = 0;
        slope = 0;
        hr_set = 0;
        sys_set = 0;
        dias_set = 0;
        last_measured_x = -1;
        frame = 0;                              /*      The data frame #  */
        end_frame = YES;                        /* "End of frame" flag   */
        dac_v = 0;                              /* DAC voltage for motor */
        last_measured = -1;
        art_press = 0;                          /*  The most recent art_p[]          */
        a_p_index = 0;                          /*  Points to the next art_p loc     */
        cuf_press = 0                           /*  The most recent cuff_p[]         */
```

```
        c_p_index = 0;                          /*  The next cuff_p datum to be entd */
        vein_press = 0;
        v_p_index = 0;
        core_temp = 0;
        c_t_index = 0;
        skin_temp = 0;
        s_t_index = 0;
        bp_ind = 0;
        motor_v = 0;                            /*  The motor velocity in rpm        */

        das8_state = 0;                         /*  Contents of the DAS-8 control register*/
        inc_dec = 0;
        ch = 0;
        newkey = 0;
        freeze = -1;
        lock = 1;

        for( i=0; i <= max_points; i++ ) {
                for( j=0; j <= 9; j++ ) {
                        data[i][j] = 0;
                }
        }
        for( i=0; i <= max_points; i++ ) {
                data[i][di] = 999;
        }

        setscreen(&page2);
        check_settings = 2;
        set_up_vs_settings_menu(check_settings);
query:
        set_up_vs_query();

        display(&page2);
        obtain_vs_settings();
        if( exit_flag == 11 ) goto out_loop;
        setcolor( &blue );
        clr();
        check_settings = 0;
        set_up_vs_settings_menu(check_settings);
        while( !kbhit() );
        ch = getch();
        if (ch == 27) goto out_loop;
        if (ch == 121 || ch == 89) {
                answer[0] = 'Y';
                stext( answer );
        }
        else {
                answer[0] = 'N';
                stext( answer );
                calflag = 0;
                goto query;
        }

        if( calflag == 11 ) goto cal0;

/* Read constants from file. */
        if ((stream = fopen("calib.arm", "r+b")) != NULL);
        else printf("Was not able to open the file.");
        result = fscanf(stream, "%d", &k_art_p);
        result = fscanf(stream, "%d", &k_cuff_p);
        result = fscanf(stream, "%d", &k_vein_p);
```

```
result = fscanf(stream, "%d", &off_art_p);
result = fscanf(stream, "%d", &off_cuff_p);
result = fscanf(stream, "%d", &off_vein_p);
fclose(stream);

period_msec = (int) ( 60.0 / (double) hr_set * 1000.0 );
num_pts_in_period = period_msec / sample_interval;

/* Calculate the number of periods of the ecg and bp waveforms to fill the
        arrays with.  Use one, two, or three periods depending on which uses
        up the most of the time. */
flt_num_pts = (60.0*1000.0)
                / (double)(hr_set*sample_interval);
single = flt_num_pts;
doubleit = 2.0 * single;
triple = 3.0 * single;
single_leftover = (int)(fabs (single - (double) (int) single) * 1000.0);
doubleit_leftover = (int)(fabs (doubleit - (double) (int) doubleit)
                                                        * 1000.0);
triple_leftover = (int)(fabs (triple - (double) (int) triple) * 1000.0);
if (single_leftover > 500) single_leftover = 1000 - single_leftover;
if (doubleit_leftover > 500) doubleit_leftover = 1000 - doubleit_leftover;
if (triple_leftover > 500) triple_leftover = 1000 - triple_leftover;
if ((int)min(single_leftover,doubleit_leftover) == single_leftover)
        if ((int)min(single_leftover,triple_leftover) == single_leftover)
                multiplier = 1;
        else
                multiplier = 3;
else if ((int)min(doubleit_leftover,triple_leftover) == doubleit_leftover)
        multiplier = 2;
else multiplier = 3;

current_length = multiplier * num_pts_in_period - 1;

/* Fill the bp waveform. */
range_bp = (double) ( sys_set - dias_set );
for (k=0; k<6; k++){
        coeff_c[k] = coeff_a[k] * range_bp / 25.0;
        coeff_d[k] = coeff_b[k] * range_bp / 25.0;
}
minimum = 9999.0;
sin_interval = 360.0 / flt_num_pts;
for ( i=0, j=multiplier ; j >= 1; j--) {
        for ( ; i <= current_length && i <= MAX_LENGTH; i++) {
                angle_int = (double)i * sin_interval;
                while( angle_int >= 360.0 )
                        angle_int -= 360.0;
                angle_radians = angle_int * PI / 180.0;
                sum = 0.0;
                for ( k=0; k<6; k++){
                        temp1 = coeff_c[k]
                                        * cos( (double) (k+1) * angle_radians );
                        temp2 = coeff_d[k]
                                        * sin( (double) (k+1) * angle_radians );
                        sum += temp1 + temp2;
                }
        bp_des[i] = (int) sum;
        if (sum < minimum) minimum = sum;
        }
}
add_on = dias_set - (int) minimum;
```

```
press_map = 0.0;
minalpha = 9999;
alpha = 100;
beta = 100;
t_off = t_off / sample_interval;
for (i=0; i <= current_length && i <= MAX_LENGTH; i++) {
        bp_des[i] += acid_on;
        press_map += (double) (bp_des[i]*bp_des[i]);
        dtemp1 = (double)bp_des[i] + ((double)off_art_p / 10.0);
        dtemp1 = (2.4414 * dtemp1) / (double)k_art_p;
        dtemp0 = (-.1334*pow(dtemp1,3.0))+(1.2178*pow(dtemp1,2.0))+(-1.2401*dtemp1)+1.2732;
        if( dtemp1 > 4.900 ) dtemp1 = 4.900;
        if( dtemp0 > 4.900 ) dtemp0 = 4.900;
        bp_d1[(i + t_off) % current_length] = (int) (dtemp1 / 0.001221);
        bp_d0[i] = (int) (dtemp0 / 0.001221);
        if( bp_d0[i] < minalpha ) minalpha = bp_d0[i];
        bp_des[i] = bp_des[i] * 10;
}
press_map = (sqrt(press_map))/(sqrt((double)(current_length)));
for (i=0; i <= current_length && i <= MAX_LENGTH; i++) {
        bp_d0[i] = bp_d0[i] - minalpha;
}

for( i=0; i <= MAX_LENGTH; i++ ) {
        ecg_des[i] = 0;
}

qrs = num_pts_in_period / 3;
ecg_des[qrs] = 50;
for( i=(qrs-1); i >= (qrs-6) && i >= 0; i-- ) {
        ecg_des[i] = ecg_coeff_a[qrs-1-i];
}
for( i=(qrs+1); i <= (qrs+9) && i <= current_length && i <= MAX_LENGTH; i++ ) {
        ecg_des[i] = ecg_coeff_b[i-qrs-1];
}

qrs += num_pts_in_period;
ecg_des[qrs] = 50;
for( i=(qrs-1); i >= (qrs-6) && i >= 0; i-- ) {
        ecg_des[i] = ecg_coeff_a[qrs-1-i];
}
for( i=(qrs+1); i <= (qrs+9) && i <= current_length && i <= MAX_LENGTH; i++ ) {
        ecg_des[i] = ecg_coeff_b[i-qrs-1];
}

qrs += num_pts_in_period;
ecg_des[qrs] = 50;
for( i=(qrs-1); i >= (qrs-6) && i >= 0; i-- ) {
        ecg_des[i] = ecg_coeff_a[qrs-1-i];
}
for( i=(qrs+1); i <= (qrs+9) && i <= current_length && i <= MAX_LENGTH; i++ ) {
        ecg_des[i] = ecg_coeff_b[i-qrs-1];
}

setscreen(&page1);
set_up_manometer_disp();
set_up_icu_disp();
set_up_process_disp();
set_up_options_bar();
update_header(header);
```

```
        display(&page1);

        most_recent_data = MAX_MEASURED - 1;
        data[0][gr] = graph = 1;
        curs_pos = 3;

/*   Start 1kHz system clock for the interrupt routine:  */

   i = set_timer( 2, 3, 5E6, 1E3 );       /* tmr 2,md 3,3MHz in,1kHz out */
   if( i < 0 )
      prir f("*** ERROR in programming 8254\n");

/*   Set interrupt frequency for the interrupt routine:    */

   i = set_timer( 1, 3, 1E3, INT_FREQ );  /* t1,m3,1k in,INT_FREQ out */
   if( i < 0 )
      printf("*** ERROR in programming 8254\n");

   das8_state = enable_ints( das8_state );        /* En ints on DAS-8 board */
   set_imr( INT_CHAN, ON );                       /*  Clear the relevent imr mask bit */

   /* Install the isr for the interrupt routine: ( C-TOOLS+ )    */
   isinstal(INT_TYPE, arm_isr, isr_id, &isr_ctrl, isr_stack, STACKSIZE,
                        NSTKS);

   das8_state = das8_state | D1_OUT_HI;   /*  Enable interrupt timer    */
   outp( DAS8_CONTROL, das8_state );

        for ( loop_counter = 0, loop_cntr = 0, most_recent_plotted = MAX_MEASURED - 1; ch != 'q';
                        loop_counter++){

                loop_cntr = ( loop_cntr + 1 ) % 1000;
                /* Wait for first set of data and for new data. */
                while (last_measured_x == last_measured);

                last_measured_x = last_measured;

                arter = data[last_measured_x][ar]/10;
                cuff = data[last_measured_x][cu]/10;
                venous = data[last_measured_x][ve]/10;
                pps = data[last_measured_x][pp];
                skin = data[last_measured_x][sk]/10;
                core = data[last_measured_x][co]/10;

                if (major_error) {

                        if ((stream = fopen("errors.run", "a")) != NULL);
                        else printf("Was not able to open the file.");
                        fprintf(stream, "%d\t%d\n", num_errors, error_num);
                        for (i = last_measured_x;
                                i >= (MAX_MEASURED + last_measured_x - 10) % MAX_MEASURED;
                                i = (MAX_MEASURED + i - 1) % MAX_MEASURED){
                                fprintf(stream, "%d\t", i);
                                for (j = 0; j <=9; j++){
                                        fprintf(stream, "%d\t", data[i][j]);
                                }
                                fputs("\n", stream);
                        }
                        fputs("\n", stream);
                        fclose(stream);
```

```
                utsound(high_pitch, long_duration);
                das8_state = das8_state & ~D1_OUT_HI;     /* Disable interrupt timer */
                outp( DAS8_CONTROL, das8_state );

                /* Closing commands for the interrupt routine. */
                outp(DAC0_HI,0x32);
                outp(DAC0_LO,0);
                outp(DAC1_HI,0xFF);
                outp(DAC1_LO,0xF0);

                /* Restore the old interrupt vector.                    */
                isputvec (INT_TYPE, pisr_ctrl->prev_vec);

                set_imr( INT_CHAN, OFF );           /*  set the relevent imr mask bit */

                setscreen( &page2 );
                setcolor( &red );
                bar(&out_x1, &out_y1, &out_x2, &out_y2);
                setcolor( &white );
                bar(&in_x1, &in_y1, &in_x2, &in_y2);
                setstclr( &red, &red );
                movtcurabs( &messg_x, &messg_y );
                setstext( &message_height, &aspect, &path );
                stext( pmessage );
                setstclr( &white, &white );
                movtcurabs( &term_x, &term_y );
                setstext( &big_height, &aspect, &path );
                stext( terminate );
                movtcurabs( &quest_x, &quest_y );
                setstext( &question_height, &aspect, &path );
                stext( start_over );
                deltcur();
                display( &page2 );

                while( !kbhit() );
                ch = getch();
                if (ch == 121 || ch == 89) {
                        answer[0] = 'Y';
                        stext( answer );
                }
                else {
                        answer[0] = 'N';
                        stext( answer );
                        goto close_out;
                }

                setscreen( &page1 );
                delbox();
                main_restart = 1;
                goto start;
        }

        if (basic_error) {

                if ((stream = fopen("errors.run", "a")) != NULL);
                else printf("Was not able to open the file.");
                fprintf(stream, "%d\t%d\n", num_errors, error_num);
                for (i = last_measured_x;
                        i >= last_measured_x - 10; i--){
                        fprintf(stream, "%d\t", i);
                        for (j = 0; j <=9; j++){
```

```
                        fprintf(stream, "%d\t", data[i][j]);
                }
                fputs("\n", stream);
        }
        fputs("\n", stream);
        fclose(stream);

        basic_error = 0;
        message_on = 1;
        num_points_msg_on = 0;
        utsound(mid_pitch, mid_duration);
}
if (message_on) {
        setcolor(&bright_blue);
        bar( &error_x1, &error_y1, &error_x2, &error_y2 );
        if (on_off_toggle == ON) {
                movtcurabs(&error_x1, &error_y1);
                setstclr(&red, &red);
                setstext(&error_height, &aspect, &path);
                stext( pmessage );
                deltcur();
                on_off_toggle = OFF;
        }
        else on_off_toggle = ON;

        num_points_msg_on += (last_measured_x
                + MAX_MEASURED - most_recent_plotted)
                % MAX_MEASURED;

        if (num_points_msg_on > (20 * (int) INT_FREQ) ) {
                num_points_msg_on = 0;
                setcolor(&bright_blue);
                bar( &error_x1, &error_y1, &error_x2, &error_y2 );
                message_on = 0;
        }
}

data[last_measured_x][gr] = graph;
update_manometer( (float) cuff );
update = loop_cntr % (2 * GRAPHS_EVERY_X);
update_icu( update, freeze, lock, inc_dec);
update_process( pps, cuff, venous, core, skin);
update_options(curs_pos);
if (new_header == 1) update_header(header);
most_recent_plotted = last_measured_x;

if ( kbhit() ){
        newkey = 1;
        ch = getch();
        if (ch == '0') {
                ch = getch();
        }
}

inc_dec = 0;

if (ch == 13 && newkey) ;
else
if (ch >= 59 && ch <= 66 && newkey) curs_pos = ch - 58;
else
if (ch >= 49 && ch <= 56 && newkey) curs_pos = ch - 48;
```

```
else
if ((ch == 72 || ch == 75) && newkey ) {
        curs_pos--;
        newkey = 0;
}
else
if ((ch == 77 || ch == 80) && newkey) {
        curs_pos++;
        newkey = 0;
}
else goto end_loop;

if (curs_pos >= 9) curs_pos = 1;
else
if (curs_pos <= 0) curs_pos = 8;

if (curs_pos == 1 && newkey) {
        freeze *= -1;
        frozen = 0;
        newkey = 0;
}
if (curs_pos == 2 && newkey) {
        lock *= -1;
        newkey = 0;
}
if (curs_pos == 3 && newkey){
        inc_dec = -1;
        at_up_limit = 0;
        if (at_low_limit) utsound( low_pitch, short_duration );
        newkey = 0;
}
if (curs_pos == 4 && newkey){
        inc_dec = 1;
        at_low_limit = 0;
        if (at_up_limit) utsound( low_pitch, short_duration );
        newkey = 0;
}
if (curs_pos == 5 && newkey){
        newkey = 0;
        check_settings = 1;
        setscreen(&page2);
        set_up_vs_settings_menu(check_settings);
        display(&page2);
        while ( !kbhit() );
        ch = getch();
        if (ch == 110 || ch == 78) {
                answer[0] = 'N';
                stext( answer );

                das8_state = das8_state & ~D1_OUT_HI;     /* Disable interrupt timer */
                outp( DAS8_CONTROL, das8_state );

                /* Closing commands for the interrupt routine. */
                outp(DAC0_HI,0x32);
                outp(DAC0_LO,0);
                outp(DAC1_HI,0xFF);
                outp(DAC1_LO,0xF0);

                /* Restore the old interrupt vector.                */
                isputvec (INT_TYPE, pisr_ctrl->prev_vec);
```

```
                        set_imr( INT_CHAN, OFF );          /*  set the relevent imr mask bit */
                        setscreen( &page1 );
                        delbox();
                        main_restart = 1;
                        goto start;
                }
                if (ch == 121 || ch == 89) {
                        answer[0] = 'Y';
                        stext( answer);
                }
                setscreen(&page1);
                display(&page1);
        }
        if (curs_pos == 6 && newkey){
                graph++;
                if (graph >= 5)
                        graph = 1;
                newkey = 0;
        }
        if (curs_pos == 7 && newkey){
                newkey = 0;
                help_system();
        }
        if (curs_pos == 8 && newkey){
                ch = 'q';
        }
end_loop:
        ;
        }

out_loop:

   das8_state = das8_state & ~D1_OUT_HI;          /* Disable interrupt timer */
   outp( DAS8_CONTROL, das8_state );

/* Closing commands for the interrupt routine. */
   outp(DAC0_HI,0x32);
   outp(DAC0_LO,0);
   outp(DAC1_HI,0xFF);
   outp(DAC1_LO,0xF0);

              /* Restore the old interrupt vector.                */
    isputvec (INT_TYPE, pisr_ctrl->prev_vec);

   set_imr( INT_CHAN, OFF );        /*  set the relevent imr mask bit */


/* Close halo graphics. */

close_out:
        ;
        closegraphics();
        goto qed;

cal0:
        ;

        outp(DAC0_HI,0);
        outp(DAC0_LO,0);

/* Read constants from file. */
```

```
        if ((stream = fopen("calib.arm", "r+b")) != NULL);
        else printf("Was not able to open the file.");
        result = fscanf(stream, "%d", &k_art_p);
        result = fscanf(stream, "%d", &k_cuff_p);
        result = fscanf(stream, "%d", &k_vein_p);
        result = fscanf(stream, "%d", &off_art_p);
        result = fscanf(stream, "%d", &off_cuff_p);
        result = fscanf(stream, "%d", &off_vein_p);
        fclose(stream);

        setcolor( &blue );
        clr();
        setstclr( &white, &white );
        setstext( &cal_height, &aspect, &path );
        movtcurabs( &cal_x, &cal1_y );
        stext( cal1_msg );
        movtcurabs( &cal_x, &cal2_y );
        stext( cal2_msg );
        movtcurabs( &cal_x, &cal3_y );
        stext( cal3_msg );
        movtcurabs( &cal_x, &cal4_y );
        stext( cal4_msg );
        movtcurabs( &cal_x, &cal5_y );
        stext( cal5_msg );
        movtcurabs( &cal_x, &cal6_y );
        stext( cal6_msg );
        movtcurabs( &cal_x, &cal7_y );
        stext( cal7_msg );
        movtcurabs( &cal_x, &cal8_y );
        stext( cal8_msg );
        movtcurabs( &cal_x, &cal9_y );
        stext( cal9_msg );
        setstclr( &yellow, &yellow );
        setstext( &press_height, &aspect, &path );

        a_p_index = 0;
        c_p_index = 0;
        v_p_index = 0;
        last_averaged = 0;
        last_measured_x = 0;
        last_measured = 0;

/*   Start 1kHz system clock for the interrupt routine:  */

   i = set_timer( 2, 3, 5E6, 1E3 );       /* tmr 2,md 3,3MHz in,1kHz out */
   if( i < 0 )
      printf("*** ERROR in programming 8254\n");

/*   Set interrupt frequency for the interrupt routine:    */

   i = set_timer( 1, 3, 1E3, INT_FREQ );  /* t1,m3,1k in,INT_FREQ out */
   if( i < 0 )
      printf("*** ERROR in programming 8254\n");

   das8_state = enable_ints( das8_state );        /* En ints on DAS-8 board */
   set_imr( INT_CHAN, ON );                       /*  Clear the relevent imr mask bit */

   /* Install the ' - for the interrupt routine: ( C-TOOLS+ )    */
   isinstal(INT_TYPE, arm_isr, isr_id, &isr_ctrl, isr_stack, STACKSIZE,
                        NSTKS);
```

```
   das8_state = das8_state | D1_OUT_HI;  /*  Enable interrupt timer    */
   outp( DAS8_CONTROL, das8_state );

        newkey = 0;
cal2:
        ;
        if ( kbhit() ){
                newkey = 1;
                ch = getch();
                if (ch == '0') {
                        ch = getch();
                }
        }

        if( newkey == 1 && ( (ch == 'e') || (ch == 'E') ) ) {
                newkey = 0;
                calflag = 0;
                goto cal99;
        }

        if( newkey == 1 && ( (ch == 'l') || (ch == 'L') ) ) {
                newkey = 0;
                calflag = 12;
                last_averaged = (last_measured + 11) % MAX_MEASURED;
                while( last_measured != last_averaged );
                last_measured_x = (last_measured+MAX_MEASURED-1) % MAX_MEASURED;
                press_set[0] = 0.0;
                press_set[1] = 0.0;
                press_set[2] = 0.0;
                for( i = 0; i < 10; i++) {
                        press_set[0] += (double) data[(last_measured_x+MAX_MEASURED-i) % MAX_MEASURED][ar];
                        press_set[1] += (double) data[(last_measured_x+MAX_MEASURED-i) % MAX_MEASURED][cu];
                        press_set[2] += (double) data[(last_measured_x+MAX_MEASURED-i) % MAX_MEASURED][ve];
                }
                art_cal = press_set[0] / 10.0;
                cuf_cal = press_set[1] / 10.0;
                ven_cal = press_set[2] / 10.0;
        }

        if( newkey == 1 && ( (ch == 'h') || (ch == 'H') ) ) {
                newkey = 0;
                if( calflag == 12 ) {
                        last_averaged = (last_measured + 11) % MAX_MEASURED;
                        while( last_measured != last_averaged );
                        last_measured_x = (last_measured+MAX_MEASURED-1) % MAX_MEASURED;
                        press_set[0] = 0.0;
                        press_set[1] = 0.0;
                        press_set[2] = 0.0;
                        for( i = 0; i < 10; i++) {
                                press_set[0]    +=    (double)    data[(last_measured_x+MAX_MEASURED-i)    %
MAX_MEASURED][ar];
                                press_set[1]    +=    (double)    data[(last_measured_x+MAX_MEASURED-i)    %
MAX_MEASURED][cu];
                                press_set[2]    +=    (double)    data[(last_measured_x+MAX_MEASURED-i)    %
MAX_MEASURED][ve];
                        }
                        press_set[0] = press_set[0] / 10.0;
                        press_set[1] = press_set[1] / 10.0;
                        press_set[2] = press_set[2] / 10.0;
                        k_art_p = (int) ( 250000.0 / (press_set[0] - art_cal) );
                        k_cuff_p = (int) ( 250000.0 / (press_set[1] - cuf_cal) );
```

```
                        k_vein_p = (int) ( 250000.0 / (press_set[2] - ven_cal) );
                        off_art_p = (int) ((art_cal * (double)k_art_p) / 100.0);
                        off_cuff_p = (int) ((cuf_cal * (double)k_cuff_p) / 100.0);
                        off_vein_p = (int) ((ven_cal * (double)k_vein_p) / 100.0);
                        if ((stream = fopen("calib.arm", "w+b")) != NULL);
                        else printf("Was not able to open the file.");
                        fprintf(stream, "%d\t", k_art_p);
                        fprintf(stream, "%d\t", k_cuff_p);
                        fprintf(stream, "%d\t", k_vein_p);
                        fprintf(stream, "%d\t", off_art_p);
                        fprintf(stream, "%d\t", off_cuff_p);
                        fprintf(stream, "%d\t", off_vein_p);
                        fclose(stream);
                        calflag = 11;
                }
        }

        art_pres_ten = art_press / 10;
        cuf_pres_ten = cuf_press / 10;
        vein_pres_ten = vein_press / 10;
        if( (last_measured % (int) INT_FREQ) == 0 ) {
                bar( &calb1_x, &calb1_y, &calb2_x, &calb2_y );
                movtcurabs( &calap_x, &calp_y );
                itoa( art_pres_ten, pressure, 10 );
                stext( pressure );
                movtcurabs( &calcp_x, &calp_y );
                itoa( cuf_pres_ten, pressure, 10 );
                stext( pressure );
                movtcurabs( &calvp_x, &calp_y );
                itoa( vein_pres_ten, pressure, 10 );
                stext( pressure );
                deltcur();
        }
        goto cal2;

cal99:
        ;
        calflag = 0;

   das8_state = das8_state & ~D1_OUT_HI;          /* Disable interrupt timer */
   outp( DAS8_CONTROL, das8_state );

/* Closing commands for the interrupt routine. */
   outp(DAC0_HI,0x32);
   outp(DAC0_LO,0);
   outp(DAC1_HI,0xFF);
   outp(DAC1_LO,0xF0);

              /* Restore the old interrupt vector.                */
    isputvec (INT_TYPE, pisr_ctrl->prev_vec);

   set_imr( INT_CHAN, OFF );        /*  set the relevent imr mask bit */
        setscreen( &page1 );
        delbox();
        main_restart = 1;
        goto start;

qed:
        ;
}
```

```
int matherr(struct exception *x)
{
        static char matherr_message[] = { " Math error: " };

        float error_x1 = (float) 700.0, error_y1 = (float) 765.0;
        float error_x2 = (float) 975.0, error_y2 = (float) 815.0;
        float error_y3 = (float) 768.0;
        float error_height = (float) 20.0;

        int i, j;
        FILE *stream;
        int path = 0;

        error_num = 9;
        num_errors++;

        if ((stream = fopen("errors.run", "a")) != NULL);
        else printf("Was not able to open the file.");
        fprintf(stream, "%d\t%d\n", num_errors, error_num);
        fprintf(stream, "%d\t%lf\t%lf\n", x->type, x->arg1, x->arg2);
        for (i = last_measured_x;
                i >= last_measured_x - 10; i--){
                fprintf(stream, "%d\t", i);
                for (j = 0; j <=9; j++){
                        fprintf(stream, "%d\t", data[i][j]);
                }
                fputs("\n", stream);
        }
        fputs("\n", stream);
        fclose(stream);

        basic_error = 0;
        message_on = 1;
        num_points_msg_on = 0;
        utsound(mid_pitch, mid_duration);
        setcolor(&bright_blue);
        bar( &error_x1, &error_y1, &error_x2, &error_y2 );
        movtcurabs(&error_x1, &error_y3);
        setstclr(&red, &red);
        setstext(&error_height, &aspect, &path);
        pmessage = x->name;
        stext( matherr_message );
        stext( pmessage );

        return(1);
}

void set_up_vs_settings_menu(int check_settings)
{
        float out_x1 = (float) 0.0, out_y1 = (float) 1000.0;
        float out_x2 = (float) 1000.0, out_y2 = (float) 0.0;
        float in_x1 = (float) 15.0, in_y1 = (float) 975.0;
        float in_x2 = (float) 985.0, in_y2 = (float) 725.0;
        float text_x1 = (float) 25.0, text_y1 = (float) 950.0;
        float text_x2 = (float) 975.0, text_y2 = (float) 900.0;
        float lower_in_x1 = (float) 15.0, lower_in_y1 = (float) 675.0;
        float lower_in_x2 = (float) 985.0, lower_in_y2 = (float) 25.0;

        int path = 0;
        float qu_height = (float) 35.0;
        float height = (float) 40.0;
```

```
float num_height = (float) 120.0;

double hr_float, sys_float;
float header_x = (float) 25.0, header_y = (float) 900.0;
float hr_label_x = (float) 245.0, hr_label_y = (float) 750.0;
static char hr_label[] = "bpm";
float bp_label_x = (float) 825.0, bp_label_y = (float) 750.0;
static char bp_label[] = "mmHg";
float hr_x = (float) 40.0, hr_y = (float) 720.0;
char hr[3];
float sys_x = (float) 340.0, sys_y = (float) 720.0;
char sys[3];
float diag_x = (float) 540.0, diag_y = (float) 730.0;
static char diag[] = "/";
float dias_x = (float) 620.0, dias_y = (float) 720.0;
char dias[3];
float question_x = (float) 95.0, question_y = (float) 600.0;
static char question[]
        = "Accept settings and begin? (Y)es or (N)o: ";
static char press_key[]
        = "Accept settings and continue? (Y)es or (N)o: ";
static char cal_ques[]
        = "Calibrate or check transducers? (Y)es or (N)o: ";
int hatch_index = 1;

setcolor(&bright_blue);
bar(&out_x1, &out_y1, &out_x2, &out_y2);
setcolor(&blue);
bar(&in_x1, &in_y1, &in_x2, &in_y2);
bar(&lower_in_x1, &lower_in_y1, &lower_in_x2, &lower_in_y2);
setcolor(&bright_blue);
bar(&text_x1, &text_y1, &text_x2, &text_y2);

setstclr(&white, &white);

/* Draw header and labels. */
sethatchstyle(&hatch_index);
setstext(&height, &aspect, &path);
movtcurabs(&header_x, &header_y);
stext( header );
movtcurabs(&hr_label_x, &hr_label_y);
stext( hr_label );
movtcurabs(&bp_label_x, &bp_label_y);
stext( bp_label );

/* Convert vs settings from integers to character strings. */
itoa(hr_set, hr, 10);
itoa(sys_set, sys, 10);
itoa(dias_set, dias, 10);

/* Write vital signs settings. */
setstext(&num_height, &aspect, &path);
hr_float = fabs((double) hr_set) + .99999;
hr_x += ( (float) ( 2 - (int) log10( hr_float ) )
                * (float) 65.0);
movtcurabs(&hr_x, &hr_y);
stext( hr );
sys_float = fabs((double) sys_set) + .99999;
sys_x += ( (float) ( 2 - (int) log10( sys_float ) )
                * (float) 65.0);
movtcurabs(&sys_x, &sys_y);
```

```
        stext( sys );
        movtcurabs(&diag_x, &diag_y);
        stext( diag );
        movtcurabs(&dias_x, &dias_y);
        stext( dias );
        if( calflag == 11 ) {
                setstext(&qu_height, &aspect, &path);
                movtcurabs(&question_x, &question_y);
                stext( cal_ques );
                goto done_check;
        }

        if (check_settings == 1){
                setstext(&qu_height, &aspect, &path);
                movtcurabs(&question_x, &question_y);
                stext( press_key );
        }

        if (check_settings == 0){
                setstext(&qu_height, &aspect, &path);
                movtcurabs(&question_x, &question_y);
                stext( question );
        }
done_check:
                ;
        deltcur();

}

void set_up_vs_query()
{
        float out_x1 = (float) 0.0, out_y1 = (float) 700.0;
        float out_x2 = (float) 1000.0, out_y2 = (float) 00.0;
        float in_x1 = (float) 15.0, in_y1 = (float) 675.0;
        float in_x2 = (float) 985.0, in_y2 = (float) 25.0;

        float hdr_lbl_x = (float) 50.0, hdr_lbl_y = (float) 600.0;
        static char hdr_lbl[] = "Report Heading:";
        float vss_lbl_x = (float) 50.0, vss_lbl_y = (float) 450.0;
        static char vss_lbl[] = "Vital Signs Settings:";
        float hr_lbl_x = (float) 360.0, hr_lbl_y = (float) 390.0;
        static char hr_lbl[] = "HR";
        float sys_lbl_x = (float) 535.0, sys_lbl_y = (float) 390.0;
        static char sys_lbl[] = "SYS";
        float diag_lbl_x = (float) 610.0, diag_lbl_y = (float) 390.0;
        static char diag_lbl[] = "/";
        float dias_lbl_x = (float) 640.0, dias_lbl_y = (float) 390.0;
        static char dias_lbl[] = "DIAS";
        float man_lbl_x = (float) 130.0, man_lbl_y = (float) 340.0;
        static char man_lbl[] = "Manual:";
        float auto_lbl_x = (float) 210.0, auto_lbl_y = (float) 285.0;
        static char auto1_lbl[] = "1:";
        static char auto2_lbl[] = "2:";
        static char auto3_lbl[] = "3:";
        static char auto4_lbl[] = "4:";
        static char auto5_lbl[] = "5:";
        int hatch_index = 1;

        static int auto_hr[4] = {60,80,72,40};
        static int auto_sys[4] = {120,90,160,150};
        static int auto_dias[4] = {80,50,90,70};
```

```
static char calib[] = "CALIBRATION";

char hr[3];
char sys[3];
char dias[3];

float hr_x = (float) 360.0;
float sys_x = (float) 535.0;
float dias_x = (float) 640.0;

int path = 0;
float height = (float) 40.0;

setcolor(&bright_blue);
bar(&out_x1, &out_y1, &out_x2, &out_y2);
setcolor(&blue);
bar(&in_x1, &in_y1, &in_x2, &in_y2);

/* Write labels. */
setstclr(&white, &white);
sethatchstyle(&hatch_index);
setstext(&height, &aspect, &path);
movtcurabs(&hdr_lbl_x, &hdr_lbl_y);
stext( hdr_lbl );
movtcurabs(&vss_lbl_x, &vss_lbl_y);
stext( vss_lbl );
movtcurabs(&hr_lbl_x, &hr_lbl_y);
stext( hr_lbl );
movtcurabs(&sys_lbl_x, &sys_lbl_y);
stext( sys_lbl );
movtcurabs(&diag_lbl_x, &diag_lbl_y);
stext( diag_lbl );
movtcurabs(&dias_lbl_x, &dias_lbl_y);
stext( dias_lbl );
movtcurabs(&man_lbl_x, &man_lbl_y);
stext( man_lbl );
movtcurabs(&auto_lbl_x, &auto_lbl_y);
stext( auto1_lbl );
movtcurabs(&hr_x, &auto_lbl_y);
itoa(auto_hr[0], hr, 10);
stext( hr );
movtcurabs(&sys_x, &auto_lbl_y);
itoa(auto_sys[0], sys, 10);
stext( sys );
movtcurabs(&dias_x, &auto_lbl_y);
itoa(auto_dias[0], dias, 10);
stext( dias );
auto_lbl_y -= (float) 55.0;
movtcurabs(&auto_lbl_x, &auto_lbl_y);
stext( auto2_lbl );
movtcurabs(&hr_x, &auto_lbl_y);
itoa(auto_hr[1], hr, 10);
stext( hr );
movtcurabs(&sys_x, &auto_lbl_y);
itoa(auto_sys[1], sys, 10);
stext( sys );
movtcurabs(&dias_x, &auto_lbl_y);
itoa(auto_dias[1], dias, 10);
stext( dias );
auto_lbl_y -= (float) 55.0;
movtcurabs(&auto_lbl_x, &auto_lbl_y);
```

```
        stext( auto3_lbl );
        movtcurabs(&hr_x, &auto_lbl_y);
        itoa(auto_hr[2], hr, 10);
        stext( hr );
        movtcurabs(&sys_x, &auto_lbl_y);
        itoa(auto_sys[2], sys, 10);
        stext( sys );
        movtcurabs(&dias_x, &auto_lbl_y);
        itoa(auto_dias[2], dias, 10);
        stext( dias );
        auto_lbl_y -= (float) 55.0;
        movtcurabs(&auto_lbl_x, &auto_lbl_y);
        stext( auto4_lbl );
        movtcurabs(&hr_x, &auto_lbl_y);
        itoa(auto_hr[3], hr, 10);
        stext( hr );
        movtcurabs(&sys_x, &auto_lbl_y);
        itoa(auto_sys[3], sys, 10);
        stext( sys );
        movtcurabs(&dias_x, &auto_lbl_y);
        itoa(auto_dias[3], dias, 10);
        stext( dias );
        auto_lbl_y -= (float) 55.0;
        movtcurabs(&auto_lbl_x, &auto_lbl_y);
        stext( auto5_lbl );
        movtcurabs(&hr_x, &auto_lbl_y);
        stext( calib );
        deltcur();
}

void set_up_manometer_disp()
{
        int i;
        float bkgnd_x1 = (float) 0.0, bkgnd_y1 = (float) 955.0;
        float bkgnd_x2 = (float) 100.0, bkgnd_y2 = (float) 48.0;
        float manom_x1 = (float) 40.0, manom_y1 = (float) 935.0;
        float manom_x2 = (float) 60.0, manom_y2 = (float) 95.0;
        float x1, x2, y1, y2;
        float odd_align;
        float odd_base = (float) 61.0;
        float mmhg_x = (float) 20.0, mmhg_y = (float) 60.0;
        float even_align;
        float even_base = (float) 0.0;
        double i_float;
        float y_place;
        float no_change = (float) 0.0;
        char value_string[3];
        float num_height = (float) 22.0;
        float height = (float) 20.0;
        int path = 0;
        float center_x = (float) 50.0, center_y = (float) 75.0;
        float bulb_radius = (float) 16.0, angle1 = (float) 105.0;
        float angle2 = (float) 75.0;
        float top_radius = (float) 10.0;
        float top_angle1 = (float) 0.0, top_angle2 = (float) 180.0;
        float temp_y;

        x1 = manom_x1 + (float) 2.0,
        x2 = manom_x2 - (float) 2.0;
        y1 = manom_y1 - (float) 2.0;
        y2 = manom_y2 + (float) 2.0;
```

```
        setcolor(&black);
        setfont( fontname2 );
        bar(&bkgnd_x1, &bkgnd_y1, &bkgnd_x2, &bkgnd_y2);
        movabs( &center_x, &center_y );
        setcolor(&white);
        bar(&x1, &y1, &x2, &y2);
        setcolor(&dark_gray);
        fcir(&bulb_radius);
        setcolor(&white);
        arc( &bulb_radius, &angle1, &angle2 );
        movabs( &center_x, &manom_y1 );
        arc( &top_radius, &top_angle1, &top_angle2 );
        temp_y = manom_y1 + (float) 2.0;
        movabs( &center_x, &temp_y );
        flood( &white );
        box(&manom_x1, &manom_y1, &manom_x2, &manom_y2);
        setstext(&height, &aspect, &path);
        setstclr(&white, 0);
        movtcurabs(&mmhg_x, &mmhg_y);
        stext( "mmHg" );
        setstext(&rxm_height, &aspect, &path);
        setcolor(&red);
        for(i=0; i<=30; i++){
                y_place = manom_y2 - (float) 15.0
                                + (float) i * (float) 28.0;
                i_float = fabs( (double) i * 10.0 ) + .99999;
                if (i%2 == 0)   {
                        even_align = even_base +
                                ( (float) (2 - (int) log10(i_float) )
                                * (float) 7.0 );
                        movtcurabs(&even_align, &y_place);
                }
                else {
                        odd_align = odd_base +
                                ( (float) (2 - (int) log10(i_float) )
                                * (float) 4.0 );
                        movtcurabs(&odd_align, &y_place);
                }
                itoa(i*10, value_string, 10);
                stext( value_string );
        }

        setfont(fontname);
        celtcur();

}

void set_up_icu_disp()
{
        float x1 = (float) 100.0, y1 = (float) 955.0;
        float x2 = (float) 1000.0, y2 = (float) 350.0;
        float rect_x1 = (float) 700.0, rect_y1 = (float) 930.0;
        float rect_x2 = (float) 975.0, rect_y2 = (float) 725.0;
        float left_axis_x = (float) 110.0;
        float right_axis_x = (float) 650.0;
        float g1_axis_y = (float) 665.0;
        float g1_axis_length = (float) 275.0;
        float g2_axis_y = (float) 360.0;
        float g2_axis_length = (float) 275.0;
        float no_change = (float) 0.0;
```

```
float hatch_length = (float) 10.0;
float minus_hatch_length = (float) -10.0;
float left_start_x, right_start_x;

float timer_label_x = (float) 130.0;
float timer_label_y = (float) 632.0;
float timer_height = (float) 18.0;
static char timer_label[] = { '1 SECOND BETWEEN MARKERS" };
float graph_label_x = (float)410.0;
static char over_graph_label[] = { "FULL SCALE = " };
float msec_label_x = (float) 580.0;
static char msec_label[] = { "MSECS" };

float height = (float) 20.0, dig_height = (float) 30.0;
float diag_height = (float) 70.0;
int path = 0;
float bp_lbl_x1 = (float) 700.0, bp_lbl_y1 = (float) 355.0;
float bp_lbl_x2 = (float) 910.0, bp_lbl_y2 = (float) 455.0;
static char bp_lbl[] = { "mmHg" };
float hr_lbl_x = (float) 830.0, hr_lbl_y = (float) 680.0;
static char hr_lbl[] = { "HR" };
float bpm_lbl_x = (float) 910.0, bpm_lbl_y = (float) 620.0;
static char bpm_lbl[] = { "bpm" };
float sys_lbl_x = (float) 750.0, sys_lbl_y = (float) 565.0;
static char sys_lbl[] = { "SYS" };
float dias_lbl_x = (float) 870.0, dias_lbl_y = (float) 565.0;
static char dias_lbl[] = { "DIAS" };
float map_lbl_x = (float) 820.0, map_lbl_y = (float) 450.0;
static char map_lbl[] = { "MAP" };
float diag_lbl_x = (float) 819.0, diag_lbl_y = (float) 485.0;
static char diag_lbl[] = { "/" },

float date_x = (float) 705.0, date_y = (float) 730.0;
char date[9];
int hatch_index = 1;
float date_height = (float) 30.0;

float gms_x = (float) 735.0, gms_y = (float) 820.0;
float gms_height = (float)100.0;
static char gms[] = { "GMS" };

float eng_x = (float) 720.0, eng_y = (float) 820.0;
float eng_height = (float) 20.0;
static char eng_corp[] = { "Engineering Corporation" };

/* Draw background rectangle. */
setcolor(&blue);
bar(&x1, &y1, &x2, &y2);

/* Draw info rectangle. */
setcolor(&bright_blue);
bar(&rect_x1, &rect_y1, &rect_x2, &rect_y2);

/* Write company name. */
sethatchstyle(&hatch_index);
setstclr(&white, &white);
setstext(&gms_height, &aspect, &path);
movtcurabs(&gms_x, &gms_y);
stext( gms );
setstext(&eng_height, &aspect, &path);
movtcurabs(&eng_x, &eng_y);
```

```
stext( eng_corp );

/* Write date. */
setstext(&date_height, &aspect, &path);
setstclr(&white, &white);
_strdate(date);
movtcurabs(&date_x, &date_y);
stext( date );

/* Draw axes. */
setcolor(&light_blue);
left_start_x = left_axis_x + hatch_length;
right_start_x = right_axis_x + minus_hatch_length;
movabs(&left_start_x, &g1_axis_y);
lnrel(&minus_hatch_length, &no_change);
lnrel(&no_change, &g1_axis_length);
lnrel(&hatch_length, &no_change);
movabs(&right_start_x, &g1_axis_y);
lnrel(&hatch_length, &no_change);
lnrel(&no_change, &g1_axis_length);
lnrel(&minus_hatch_length, &no_change);
movabs(&left_start_x, &g2_axis_y);
lnrel(&minus_hatch_length, &no_change);
lnrel(&no_change, &g2_axis_length);
lnrel(&hatch_length, &no_change);
movabs(&right_start_x, &g2_axis_y);
lnrel(&hatch_length, &no_change);
lnrel(&no_change, &g2_axis_length);
lnrel(&minus_hatch_length, &no_change);

/* Draw timer label. */
setfont( fontname2 );
setstext(&timer_height, &aspect, &path);
movtcurabs(&timer_label_x, &timer_label_y);
setstclr(&light_blue, 0);
stext( timer_label );
movtcurabs(&graph_label_x, &timer_label_y);
stext( over_graph_label );
movtcurabs(&msec_label_x, &timer_label_y);
stext( msec_label );
setfont( fontname );

/* Draw BP graph label. */
setstext(&height, &aspect, &path);
movtcurabs(&bp_lbl_x1, &bp_lbl_y1);
setstclr(&yellow, 0);
stext( bp_lbl );

/* Draw digital display labels. */
setstext(&dig_height, &aspect, &path);
movtcurabs(&hr_lbl_x, &hr_lbl_y);
stext( hr_lbl );
movtcurabs(&sys_lbl_x, &sys_lbl_y);
stext( sys_lbl );
movtcurabs(&dias_lbl_x, &dias_lbl_y);
stext( dias_lbl );
movtcurabs(&map_lbl_x, &map_lbl_y);
stext( map_lbl );
setstext(&height, &aspect, &path);
movtcurabs(&bp_lbl_x2, &bp_lbl_y2);
stext( bp_lbl );
```

```
        movtcurabs(&bpm_lbl_x, &bpm_lbl_y);
        stext( bpm_lbl );
        setstext(&diag_height, &aspect, &path);
        movtcurabs(&diag_lbl_x, &diag_lbl_y);
        stext( diag_lbl );
        deltcur();
}

void set_up_process_disp()
{
        float x1 = (float) 100.0, y1 = (float) 350.0;
        float x2 = (float) 1000.0, y2 = (float) 48.0;

        float height = (float) 25.0;
        float mmhg_height = (float) 20.0;
        int path = 0;
        float cuff_x = (float) 409.0, cuff_y = (float) 320.0;
        static char cuff[] = { "CUFF" };
        float press_x1 = (float) 409.0, press_y1 = (float) 298.0;
        float press_x2 = (float) 460.0, press_y2 = (float) 171.0;
        static char press[] = { "PRESSURE" };
        float venous_x = (float) 460.0, venous_y = (float) 193.0;
        static char venous[] = { "VENOUS" };
        float mmhg_x1 = (float) 479.0, mmhg_y1 = (float) 265.0;
        float mmhg_x2 = (float) 530.0, mmhg_y2 = (float) 145.0;
        static char mmhg[] = { "mmHg" };
        float core_x = (float) 700.0, core_y = (float) 193.0;
        static char core[] = { "CORE" };
        float temp_x1 = (float) 700.0, temp_y1 = (float) 171.0;
        float temp_x2 = (float) 800.0, temp_y2 = (float) 289.0;
        static char temp[] = { "TEMP" };
        float skin_x = (float) 800.0, skin_y = (float) 311.0;
        static char skin[] = { "SKIN" };
        float degrees_c_x1 = (float) 755.0;
        float degrees_c_y1 = (float) 145.0;
        float degrees_c_x2 = (float) 855.0;
        float degrees_c_y2 = (float) 260.0;
        static char degrees_c[] = { "'C" };

        float cx1 = (float) 382.0, cy1 = (float) 291.0;
        float cx2 = (float) 388.0, cy2 = (float) 270.0;
        float vx1 = (float) 434.0, vy1 = (float) 145.0;
        float vx2 = (float) 439.0, vy2 = (float) 128.0;
        float radius = (float) 15.0;
        float arc_radius = (float) 11.0;
        float angle1 = (float) 20.0, angle2 = (float) 160.0;
        float cuff_gage_x = (float) 386.0;
        float cuff_gage_y = (float) 315.0;
        float venous_gage_x = (float) 437.0;
        float venous_gage_y = (float) 170.0;

        float s_therm_x = (float) 790.0, s_therm_y = (float) 270.0;
        float c_therm_x = (float) 690.0, c_therm_y = (float) 135.0;
        float no_change = (float) 0.0;
        float therm_height = (float) 50.0;
        float half_therm_width = (float) 4.0;
        float therm_radius = (float) 3.0;
        float therm_angle1 = (float) 0.0;
        float therm_angle2 = (float) 180.0;
        float end_arc, who_cares;
```

```
/* Draw background rectangle. */
setcolor(&white);
bar(&x1, &y1, &x2, &y2);

/* Draw loop outline. */
setcolor(&red);
draw_loop_outline();

/* Write labels and units. */
setstext(&height, &aspect, &path);
setstclr(&blue, &blue);
movtcurabs(&cuff_x, &cuff_y);
stext( cuff );
movtcurabs(&press_x1, &press_y1);
stext( press );
setstext(&mmhg_height, &aspect, &path);
movtcurabs(&mmhg_x1, &mmhg_y1);
stext( mmhg );
setstclr(&bright_pink, &bright_pink);
setstext(&height, &aspect, &path);
movtcurabs(&venous_x, &venous_y);
stext( venous );
movtcurabs(&press_x2, &press_y2);
stext( press );
setstext(&mmhg_height, &aspect, &path);
movtcurabs(&mmhg_x2, &mmhg_y2);
stext( mmhg );
setstclr(&red, &red);
setstext(&height, &aspect, &path);
movtcurabs(&core_x, &core_y);
stext( core );
movtcurabs(&temp_x1, &temp_y1);
stext( temp );
movtcurabs(&degrees_c_x1, &degrees_c_y1);
stext( degrees_c );
movtcurabs(&skin_x, &skin_y);
stext( skin );
movtcurabs(&temp_x2, &temp_y2);
stext( temp );
movtcurabs(&degrees_c_x2, &degrees_c_y2);
stext( degrees_c );
deltcur();

/* Draw pressure gages. */
setcolor(&bright_blue);
bar( &cx1, &cy1, &cx2, &cy2 );
movabs(&cuff_gage_x, &cuff_gage_y);
fcir( &radius );
setcolor(&black);
box( &cx1, &cy1, &cx2, &cy2 );
cir( &radius );
arc( &arc_radius, &angle1, &angle2);
setcolor(&bright_pink);
bar( &vx1, &vy1, &vx2, &vy2);
movabs(&venous_gage_x, &venous_gage_y);
fcir( &radius );
setcolor(&black);
box( &vx1, &vy1, &vx2, &vy2);
cir( &radius );
arc( &arc_radius, &angle1, &angle2);
```

```
        /* Draw the manometers. */
        setcolor( &red );
        movabs( &c_therm_x, &c_therm_y );
        lnrel( &no_change, &therm_height );
        movrel( &half_therm_width, &no_change );
        arc( &therm_radius, &therm_angle1, &therm_angle2);
        movabs( &c_therm_x, &c_therm_y );
        movrel( &half_therm_width, &no_change );
        arc( &therm_radius, &therm_angle2, &therm_angle1);
        inqarc( &who_cares, &who_cares, &end_arc, &who_cares );
        movabs( &end_arc, &c_therm_y );
        lnrel( &no_change, &therm_height );

        movabs( &s_therm_x, &s_therm_y );
        lnrel( &no_change, &therm_height );
        movrel( &half_therm_width, &no_change );
        arc( &therm_radius, &therm_angle1, &therm_angle2 );
        movabs( &s_therm_x, &s_therm_y );
        movrel( &half_therm_width, &no_change );
        arc( &therm_radius, &therm_angle2, &therm_angle1);
        inqarc( &who_cares, &who_cares, &end_arc, &who_cares );
        movabs( &end_arc, &s_therm_y );
        lnrel( &no_change, &therm_height );

}

void set_up_options_bar()
{
        float x1 = (float) 0.0, y1 = (float) 0.0;
        float x2 = (float) 1000.0, y2 = (float) 47.0;
        int path = 0;
        float height = (float) 22.0;
        float words_x = (float) 4.0, words_y = (float) 0.0;
        float funcs_x = (float) 0.0, funcs_y = (float) 19.0;
        int hatch_index = 1;
        static char func1[] = "  F1       F2        F3       ";
        static char word1[] = "FREEZE  LOCK-ZERO  DEC-SPEED";
        static char func2[] = " F4        F5         F6 ";
        static char word2[] = "  INC-SPEED  SETTINGS  CHANGE-GRAPH";
        static char func3[] = "       F7       F8";
        static char word3[] = "  HELP STOP-PROCESS";

        setcolor(&bright_blue);
        bar(&x1, &y1, &x2, &y2);
        setfont( fontname2 );
        sethatchstyle(&hatch_index);
        setstclr(&white, &white);
        setstext(&height, &aspect, &path);
        movtcurabs(&funcs_x, &funcs_y);
        stext( func1 );
        stext( func2 );
        stext( func3 );
        movtcurabs(&words_x, &words_y);
        stext( word1 );
        stext( word2 );
        stext( word3 );
        setfont( fontname );
        deltcur();
}

void update_manometer( float cuff_pressure )
```

```
{
        float div_length = (float) 14.0;
        float minus_small = (float) -5.0;
        float hatch_length = (float) 4.0;
        float up_hatch = (float) 14.0;
        float move_x, move_y;
        float no_change = (float) 0.0;
        float manom_x1 = (float) 40.0, manom_x2 = (float) 60.0;
        float manom_y1 = (float) 935.0, manom_y2 = (float) 95.0;
        float manom_range;
        float x1, x2, y1, y2;
        float max_pressure = (float) 300.0;
        static float old_cuff_press = (float)-1.0;
        static int old_max_hatch = 0;
        int i, max_hatch;
        int top_hatch, bottom_hatch;
        float height_chunk;
        static int first_time_manom = 1;

        if (manom_restart) {
                old_cuff_press = (float)-1.0;
                old_max_hatch = 0;
                first_time_manom = 1;
                manom_restart = 0;
        }

        height_chunk = (float) 2.0 * up_hatch;
        x1 = manom_x1 + (float) 4.0;
        x2 = manom_x2 - (float) 4.0;
        y1 = manom_y1 - (float) 2.0;
        y2 = manom_y2 + (float) 2.0;
        manom_range = y1 - y2;
        max_hatch = (int)(cuff_pressure / (float) 10.0);
        if (cuff_pressure > (float)300.0) cuff_pressure = (float)300.0;
        if (cuff_pressure < (float)0.0) cuff_pressure = (float)0.0;
        cuff_pressure = y2 + (cuff_pressure / max_pressure)
                                        * manom_range;
  if (first_time_manom) {
                old_cuff_press = y2;
                setcolor( &dark_gray );
                bar( &x1, &cuff_pressure, &x2, &old_cuff_press);
                top_hatch = 29;
                bottom_hatch = 0;
                first_time_manom = 0;
  }
  else {
        if ((int)cuff_pressure == (int)old_cuff_press) return;
        if (cuff_pressure < old_cuff_press) {
                setcolor( &white );
                if (cuff_pressure < y2) cuff_pressure = y2;
                bar( &x1, &old_cuff_press, &x2, &cuff_pressure );
                top_hatch = old_max_hatch;
                bottom_hatch = max_hatch;
        }
        if (cuff_pressure > old_cuff_press) {
                setcolor( &dark_gray );
                if (cuff_pressure > y1) cuff_pressure = y1;
                bar( &x1, &cuff_pressure, &x2, &old_cuff_press);
                top_hatch = max_hatch;
                bottom_hatch = old_max_hatch;
        }
```

```
  }
        setcolor( &red );
        move_x = manom_x1+(float) 2.0;
        move_y = manom_y2-(float) 1.0+((float)bottom_hatch * height_chunk);
        movabs(&move_x, &move_y);
        lnrel(&div_length, &no_change);
        move_x = (float) -1.0 * hatch_length;
        setcolor(&red);
        for(i=bottom_hatch; i<=29 && i<=top_hatch; i++){
                movrel(&minus_small, &no_change);
                movrel(&move_x, &up_hatch);
                lnrel(&hatch_length, &no_change);
                movrel(&minus_small, &no_change);
                movrel(&move_x, &up_hatch);
                lnrel(&div_length, &no_change);
        }
        old_cuff_press = cuff_pressure;
        old_max_hatch = max_hatch;
}

void update_icu(int update, int freeze, int lock, int inc_dec)
{
        int hatch_index = 1;
        int path = 0;
        float height = (float) 80.0;

        static int changed_date = 0;

        int time_over_graph, temp_recent_data, num_pts_plotted;
        float msec_x1 = (float) 527.0, msec_y1 = (float) 632.0;
        float msec_x2 = (float) 570.0, msec_y2 = (float) 652.0;
        float msec_height = (float) 20.0;
        char msec_string[5];
        int old_msec = 0;

        float hr_base_x = (float) 780.0;
        float sys_base_x = (float) 700.0;
        float dias_base_x = (float) 865.0;
        float map_base_x = (float) 780.0;
        float hr_x, hr_y = (float) 590.0;
        float sys_x, sys_y = (float) 475.0;
        float dias_x, dias_y = (float) 475.0;
        float map_x, map_y = (float) 360.0;
        float plus = (float) 10.0;
        float hr_y1 = hr_y + plus;
        float sys_y1 = sys_y + plus;
        float dias_y1 = dias_y + plus;
        float map_y1 = map_y + plus;
        float hr_x2 = (float) 910.0, hr_y2 = (float) 675.0;
        float sys_x2 = (float) 835.0, sys_y2 = (float) 560.0;
        float dias_x1 = (float) 865.0, dias_x2 = (float) 1000.0;
        float map_x2 = (float) 1000.0, map_y2 = (float) 445.0;
        char string[3];
        static int sys = 0;
        static int dias = 999;
        static int hr = 0;
        static int map = 0;
        static int old_hr = -99;
        static int old_sys = -99;
        static int old_dias = -99;
        static int old_map = -99;
```

```
double map_float, hr_float, sys_float, dias_float;

float date_height = (float) 30.0;
float time_x = (float) 860.0, time_y = (float) 730.0;
float time_x2 = (float) 975.0, time_y2 = (float) 760.0;
float date_x = (float) 705.0, date_y = (float) 730.0;
char date[9];
char time[9];
static char old_second = 'g';

float g_x1 = (float) 112.0;
int g1_y1 = 669;
float g1_y1_float = (float) 669.0;
float g_x2 = (float) 648.0;
int g1_y2 = 938;
float g1_y2_float = (float) 938.0;
float length_graph, length_plotted_graph;
float length_left, percent_of_line;
float y_difference;
int g2_y1 = 362, g2_y2 = 631;
float g2_y1_float = (float) 362.0, g2_y2_float = (float) 631.0;
float over_x1, old_over_x1;
int set = 1;

int g1_range, bp_range, g2_range;
int bp_max, bp_min;
int bp;
static int first_time_bp = 1;
float hatch_length = (float) 3.0;
float dot_length = (float) 1.0;
float no_change = (float) 0.0;
float timer_level = (float) 658.0;
float timer_base = (float) 655.0;
float timer_height = (float) 7.0;
float top_timer;
float newval;
static float g2_oldval;
static float ecg_g1_oldval;
int ecg, ecg_range;
int ecg_max = 10, ecg_min = -10;
static float cuff_g1_oldval;
static float venous_g1_oldval;
int press_range;
int press_max = 300, press_min = 0;
int cuff, venous;
static float pps_g1_oldval;
int pps, pps_range;
int pps_max = 6000, pps_min = 0;
int temp_max = 50, temp_min = 0;
int arter_max = 300, arter_min = 0;
int more_old_data, plot_point, previous_point;
int old_g1_graph = 0;
int new_point;
int difference;
int less_new;
static int num_loops = 0;
static double sum_squares = 0.0;
static int sys_array[10] = {0,0,0,0,0,0,0,0,0,0};
static int dias_array[10]
                        = {999,999,999,999,999,999,999,999,999,999};
static int map_array[10] = {0,0,0,0,0,0,0,0,0,0};
```

```
static int hr_array[10] = {0,0,0,0,0,0,0,0,0,0};
static int first_time = 1;
static int alpha_inc, beta_inc, alpha_sign, beta_sign;
static int actual_pp, desired_pp, err_pp, err_dias;
int i;
int fldsize = 3;

static int positv_slope = 1;
static int reset = 1;
static int upper_not_crossed = 1;
static int lower_not_crossed = 1;
static int avg_bp = 1;
static int end_of_period = 0;
static int end_of_period_last = 0;

int arter_hi_max = 360, arter_low_max = 330;
int temp_alrm_max = 43;
int temp_diff_max = 10;
int pps_alrm_max = 6000;
int cuff_hi_max = 360;
int cuff_low_max = 330;
static int num_points_over_max = 0;


if (icu_restart) {
        changed_date = 0;
        sys = 0;
        dias = 999;
        hr = 0;
        map = 0;
        old_hr = -99;
        old_sys = -99;
        old_dias = -99;
        old_map = -99;
        old_second = 'g';
        first_time_bp = 1;
        num_loops = 0;
        sum_squares = 0.0;
        for (i=0; i<=9; i++) {
                sys_array[i] = 0;
                dias_array[i] = 999;
                map_array[i] = 0;
                hr_array[i] = 0;
        }
        first_time = 1;
        positv_slope = 1;
        reset = 1;
        upper_not_crossed = 1;
        lower_not_crossed = 1;
        avg_bp = 1;
        end_of_period = 0;
        end_of_period_last = 0;
        num_points_over_max = 0;
        icu_restart = 0;
}

/* Draw info box.  Write in time and date. */
setcolor( &bright_blue );
sethatchstyle(&hatch_index);
setstext(&date_height, &aspect, &path);
setstclr(&white, &white);
```

```
_strtime(time);
/* Only rewrite date if it has changed. */
if (time[0]=='0' && time[1]=='0' && changed_date == 0){
        bar( &date_x, &date_y, &time_x, &time_y2 );
        _strdate(date);
        movtcurabs(&date_x, &date_y);
        stext( date );
        changed_date = 1;
}
/* Only rewrite time if it has changed. */
if (time[7] != old_second){
        bar( &time_x, &time_y, &time_x2, &time_y2 );
         movtcurabs(&time_x, &time_y);
        stext( time );
        deltcur();
}
old_second = time[7];


/* Update distance to move point across screen when change has
        been requested. */
if (inc_dec != 0) {
        over += (delta_over * inc_dec);
        if (over <= min_over) {
                over = min_over;
                at_low_limit = 1;
        }
        if (over >= max_over) {
                over = max_over;
                at_up_limit = 1;
        }
}
over_x1 = g_x2;

g1_range = g1_y2 - g1_y1;
g2_range = g2_y2 - g2_y1;

/* Set bp max and min graph values depending on whether lock
        or unlock zero option is selected. */
if (lock == 1) {
        bp_min = 0;
        bp_max = 300;
}
if (lock == -1) {
        bp_min = dias - 30;
        if (bp_min < 0) bp_min = 0;
        bp_max = sys + 30;
        if (bp_max > 300) bp_max = 300;
}
bp_range = bp_max - bp_min;


/* Update sys, dias, map and hr values. */
new_point = (most_recent_plotted + 1) % MAX_MEASURED;

while ( new_point != (last_measured_x + 1) % MAX_MEASURED) {

        /*  The first time that new_point is ever greater than zero,
                it is possible to calculate a slope (since there are at
                least two data points).  Until the slope is non-zero, do not
                process the data.  Once the slope is non_zero, begin processing
```

```
        the data and never enter this loop again. */

if (first_time) {
        if ( new_point >= 1 ) {
                less_new = new_point - 1;
                difference = data_base[less_new] - data_base[new_point];
                if (difference < -2) slope = 1;
                if (difference > 2) slope = -1;

                if (slope == 0) goto end_of_dig_loop;
                else {
                        first_time = 0;
                }
        }
}

/* Calculate whether the slope of the waveform is flat, negative, or
        positive.  Consider it flat if differences are small. */
else {
        less_new = (new_point + MAX_MEASURED - 1) % MAX_MEASURED;
        difference = data_base[less_new] - data_base[new_point];
        slope = 0;
        if (difference < -2) slope = 1;
        if (difference > 2) slope = -1;
}

/* For every point check for values over allowable limits. */

if (data[new_point][ar] > arter_low_max*10) {
        error_num = 6;
        num_errors++;
        pmessage = &lo_arter_message[0];
        basic_error = 1;
}
if (data[new_point][pp] > pps_alrm_max) {
        num_points_over_max++;
        if (num_points_over_max > (30 * (int) INT_FREQ) ) {
                error_num = 7;
                num_errors++;
                num_points_over_max = 0;
                pmessage = &pps_message[0];
                basic_error = 1;
        }
}
else num_points_over_max = 0;

if (data[new_point][cu] > cuff_low_max*10) {
        error_num = 8;
        num_errors++;
        pmessage = &lo_cuff_message[0];
        basic_error = 1;
}


if (data[new_point][ar] > arter_hi_max*10) {
        error_num = 1;
        num_errors++;
        major_error = 1;
        pmessage = &hi_arter_message[0];
}
if (data[new_point][cu] > cuff_hi_max*10) {
```

```
                error_num = 2;
                num_errors++;
                major_error = 1;
                pmessage = &hi_cuff_message[0];
        }
        if (data[new_point][co] > temp_alrm_max*10) {
                error_num = 3;
                num_errors++;
                major_error = 1;
                pmessage = &core_message[0];
        }
        if (data[new_point][sk] > temp_alrm_max*10) {
                error_num = 4;
                num_errors++;
                major_error = 1;
                pmessage = &skin_message[0];
        }


/*  For every point, update number of points, sum of squares, and when
        appropriate, systolic or diastolic estimates.  */

        num_pts++;

        /* If highest value so far, consider it systolic. */
        if ( data_base[new_point] > sys_array[num_loops]*10 )
                sys_array[num_loops] = data_base[new_point]/10;

        /* If lowest value so far, consider it diastolic. */
        if ( data_base[new_point] < dias_array[num_loops]*10 )
                dias_array[num_loops] = data_base[new_point]/10;

        sum_squares += (double) (data_base[new_point]/10)
                                        * (double) (data_base[new_point]/10 );

/*  Determine whether have reached end of a period by keeping
        track of when waveform passes through a range (at 3/4 of
        the distance up between systolic and diastolic) with
        the same slope as it had at the end of the last period. */

          avg_bp = ((3* old_sys) + old_dias) / 4;

          if (slope == positv_slope) {

                if (lower_not_crossed) {
                        if ( data_base[new_point] > ((10*avg_bp) - 20) ) {
                           lower_not_crossed = 0;
                           if ( data_base[new_point] > ((10*avg_bp) + 20 ) ) {
                                        upper_not_crossed = 0;
                                        end_of_period++;
                                        reset = 0;
                                }
                        }
                }
                else {
                        if (upper_not_crossed) {
                                if (data_base[new_point] > ((10*avg_bp) + 20) ) {
                                        upper_not_crossed = 0;
                                        end_of_period++;
                                        reset = 0;
                                }
```

```
                        }
                }

        /* Reset the "not_crossed" markers for next time slope becomes
                positive. */
          }
          if ( (slope != positv_slope) && !reset ) {
                upper_not_crossed = 1;
                lower_not_crossed = 1;
                reset = 1;
        }


/*  Only calculate and update digital bp values at end of a period or when
        num_pts is 50% greater than the num_pts_in_period. */

        if ( (num_pts == (int) ((double)num_pts_in_period * 1.5) ) ||
                (end_of_period != end_of_period_last) ) {

                if( num_pts < (int) ((double)num_pts_in_period * .9) ) {
                        upper_not_crossed = 1;
                        lower_not_crossed = 1;
                        reset = 1;
                        end_of_period--;
                        goto keep_going;
                }

                if (end_of_period != end_of_period_last ) {
                        hr_array[num_loops] = (int)
                                ((60.0 / (double)num_pts)
                                * (1000.0 / (double)sample_interval));

                        map_array[num_loops] = (int)( sqrt(sum_squares) / sqrt((double)num_pts));
                }

                num_pts = 0;
                num_loops++;

                 if (num_loops >= UPDATE_X_LOOPS){
                  num_loops = 0;

                  sys = 0;
                  dias = 0;
                  map = 0;
                  hr = 0;


                  for(i=0; i < UPDATE_X_LOOPS; i++){
                        sys += sys_array[i];
                        sys_array[i] = 0;
                        dias += dias_array[i];
                        dias_array[i] = 999;
                        map += map_array[i];
                        map_array[i] = 0;
                        hr += hr_array[i];
                        hr_array[i] = 0;
                  }
                  sys = sys / UPDATE_X_LOOPS;
                  if( sys < 0 ) sys = 0;
                  if( sys > 300 ) sys = 999;
                  sys_valve = sys;
```

```
dias = dias / UPDATE_X_LOOPS;
if( dias < 0 ) dias = 0;
if( dias > 300 ) dias = 999;
dias_valve = dias;
map = map / UPDATE_X_LOOPS;
if( map < 0 ) map = 0;
if( map > 300 ) map = 999;
hr = hr / UPDATE_X_LOOPS;
if( hr < 0 ) hr = 0;
if( hr > 180 ) hr = 999;
if( sys < dias ) {
      sys = 0;
      dias = 0;
}



/* Write in large bp numbers if they have
      changed.  For hr and map, only write values
      if end of period has been reached and is equal
      to update_x_loops. */

  setcolor( &blue );
  setstext(&height, &aspect, &path);
  setstclr(&yellow, &yellow);
  if (end_of_period != UPDATE_X_LOOPS) {
       bar( &hr_base_x, &hr_y1, &hr_x2, &hr_y2 );
       bar( &map_base_x, &map_y1, &map_x2, &map_y2 );
  }
  if (hr != old_hr && end_of_period == UPDATE_X_LOOPS) {
       bar( &hr_base_x, &hr_y1, &hr_x2, &hr_y2 );
       hr_float = fabs((double) hr) + .99999;
       hr_x = hr_base_x + ( (float) ( 2 * (int) log10( hr_float ) )
                     * (float) 35.0);
       movtcurabs(&hr_x, &hr_y);
       itoa( hr, string, 10 );
       stext( string );
  }
  if (sys != old_sys) {
       bar( &sys_base_x, &sys_y1, &sys_x2, &sys_y2 );
       sys_float = fabs((double) sys) + .99999;
       sys_x = sys_base_x + ( (float) ( 2 * (int) log10( sys_float ) )
                     * (float) 35.0);
       movtcurabs(&sys_x, &sys_y);
       itoa( sys, string, 10 );
       stext( string );
  }
  if (dias != old_dias) {
       bar( &dias_base_x, &dias_y1, &dias_x2, &sys_y2 );
       dias_float = fabs((double) dias) + .99999;
       dias_x = dias_base_x + ( (float)
                     ( 2 * (int) log10( dias_float ) )
                     * (float) 35.0);
       movtcurabs(&dias_x, &dias_y);
       itoa( dias, string, 10 );
       stext( string );
  }
  if (map != old_map && end_of_period == UPDATE_X_LOOPS) {
       bar( &map_base_x, &map_y1, &map_x2, &map_y2 );
       map_float = fabs((double) map) + .99999;
       map_x = map_base_x + ( (float) ( 2 * (int) log10( map_float ) )
```

```
                                        * (float) 35.0);
                        movtcurabs(&map_x, &map_y);
                        itoa( map, string, 10 );
                        stext( string );
                    }

                    deltcur();
                    old_hr = hr;
                    old_sys = sys;
                    old_dias = dias;
                    old_map = map;

/*      adjust coefficients alpha and beta to bring actual pressures to set point */

                    actual_pp = sys - dias;
                    desired_pp = sys_set - dias_set;
                    alpha_inc = 50;
                    beta_inc = 50;
                    alpha_sign = 1;
                    beta_sign = 1;
                    err_pp = abs( actual_pp - desired_pp );
                    if( err_pp <= 25 ) alpha_inc = 20;
                    if( err_pp <= 15 ) alpha_inc = 10;
                    if( err_pp <= 10 ) alpha_inc = 5;
                    if( err_pp <= 5 ) alpha_inc = 1;
                    if( err_pp <= 1 ) alpha_inc = 0;
                    if( actual_pp > desired_pp ) alpha_sign = -1;
                    alpha = alpha + (alpha_sign * alpha_inc);
                    if( alpha > 1000 ) alpha = 1000;
                    if( alpha < 10 ) alpha = 10;
                    err_dias = abs( dias - dias_set );
                    if( err_dias <= 25 ) beta_inc = 20;
                    if( err_dias <= 15 ) beta_inc = 10;
                    if( err_dias <= 10 ) beta_inc = 5;
                    if( err_dias <= 5 ) beta_inc = 1;
                    if( err_dias <= 1 ) beta_inc = 0;
                    if( dias > dias_set ) beta_sign = -1;
                    beta = beta + (beta_sign * beta_inc);
                    if( beta > 1000 ) beta = 1000;
                    if( beta < 1 ) beta = 1;
                    if( beta > ( alpha * 2 ) ) beta = ( alpha * 2 );
                    end_of_period = 0;
                  }
                end_of_period_last = end_of_period;
                sum_squares = 0.0;
keep_going:
            ;
            }


        /*  For every point, check whether values are above or below max or
                min values.  */

            if (data[new_point][ar]/10 > bp_max) data[new_point][ar] = bp_max*10;
            if (data[new_point][ar]/10 < bp_min) data[new_point][ar] = bp_min*10;
            if (data[new_point][ec]/10 > ecg_max) data[new_point][ec] = ecg_max*10;
            if (data[new_point][ec]/10 < ecg_min) data[new_point][ec] = ecg_min*10;
            if (data[new_point][cu]/10 > press_max) data[new_point][cu] = press_max*10;
            if (data[new_point][cu]/10 < press_min) data[new_point][cu] = press_min*10;
            if (data[new_point][pp] > pps_max) data[new_point][pp] = pps_max;
            if (data[new_point][pp] < pps_min) data[new_point][pp] = pps_min;
```

```
                if (data[new_point][ve]/10 > press_max) data[new_point][ve] = press_max*10;
                if (data[new_point][ve]/10 < press_min) data[new_point][ve] = press_min*10;
                if (data[new_point][co]/10 > temp_max) data[new_point][ar] = temp_max*10;
                if (data[new_point][co]/10 < temp_min) data[new_point][ar] = temp_min*10;
                if (data[new_point][sk]/10 > temp_max) data[new_point][ar] = temp_max*10;
                if (data[new_point][sk]/10 < temp_min) data[new_point][ar] = temp_min*10;

end_of_dig_loop:
                new_point = ++new_point % MAX_MEASURED;
        }


        new_point = last_measured_x;

   while ( new_point != most_recent_data ) {

        data[new_point][sk]++;

        data[new_point][di] = over;
        data[new_point][gr] = data[last_measured_x][gr];

/* Calculate y graph positions for newest set of data. */
        ecg = data[new_point][ec]/10;
        ecg_range = ecg_max - ecg_min;
        data[new_point][ec] = g1_y1 + (int) ( (float)(ecg - ecg_min)
                                        * (float)g1_range / (float)ecg_range );
        cuff = data[new_point][cu]/10;
        press_range = press_max - press_min;
        data[new_point][cu] = g1_y1 + (int) ( (cuff - press_min)
                                        * (float)g1_range / (float)press_range );
        venous = data[new_point][ve]/10;
        data[new_point][ve] = g1_y1 + (int) ( (venous - press_min)
                                        * (float)g1_range / (float)press_range );
        pps = data[new_point][pp];
        pps_range = pps_max - pps_min;
        data[new_point][pp] = g1_y1 + (int) ( (pps - pps_min)
                                        * (float)g1_range / (float)pps_range );
        data[new_point][ra] = bp_range;
        bp = data[new_point][ar]/10;
        data[new_point][ar] = g2_y1 + (int)( (bp - bp_min) *
                                        (float)g2_range / (float)bp_range );
        data[new_point][sk] = last_measured;
        data[new_point][co] = last_measured_x;

        new_point = (MAX_MEASURED + --new_point) % MAX_MEASURED;
   }
        most_recent_data = last_measured_x;

/*  If graphs are already in freeze state, do not update graphs
                at all.  Skip over the rest of the subroutine. */
if ( !frozen ) {

        setcolor( &blue );
        /* If just freezing graph, draw navy background to prepare
                to update both graphs. */
        if (freeze == 1) setcolor( &navy );
        /* Only draw over old graph if a new graph is to be drawn. */

        if (update == GRAPHS_EVERY_X || freeze == 1) {
                        bar(&g_x1, &g1_y1_float, &g_x2, &g1_y2_float);
                        bar(&g_x1, &g2_y1_float, &g_x2, &g2_y2_float);
```

```
                        top_timer = timer_base + timer_height;
                        setcolor( &blue );
                        bar( &g_x2, &timer_base, &g_x1, &top_timer );
        }

        /* Prepare to keep track of length of graph plotted and
                whether there is more data left to plot in data array as
                moving backwards through array. */
        length_graph = g_x2 - g_x1 + (float) 1.0;

        draw_graph_labels( data[most_recent_data][gr],
                                                bp_min, bp_max);

/* Loop to draw top graphs. Starts with most recent data point at the
        right of the screen and continues to draw every previous point
        until there is no more old data or reach the end of the graph.
        Only draws graph when have just frozen or it is time.
*/

  if (update == GRAPHS_EVERY_X || freeze == 1) {
        more_old_data = 1;
        length_plotted_graph = (float) 0.0;

        setcolor(&light_blue);
        movabs( &g_x2, &timer_level );
        lnabs( &g_x1, &timer_level );

        for( plot_point = most_recent_data;
                more_old_data && length_plotted_graph < length_graph;
                plot_point--){

                plot_point = (MAX_MEASURED + plot_point) % MAX_MEASURED;
                previous_point = (MAX_MEASURED + plot_point - 1) % MAX_MEASURED;

                if (data[previous_point][di] == 999) {
                        more_old_data = 0;
                        goto end_lp1;
                }
                length_plotted_graph += (float) data[plot_point][di];

                if ((plot_point % (int) INT_FREQ) == 0) {
                        over_x1 = g_x2 - length_plotted_graph;
                        if (over_x1 < g_x1) goto end_lp1;
                        movabs( &over_x1, &timer_base );
                        lnrel( &no_change, &timer_height );
                }
end_lp1:
        ;
        }


        more_old_data = 1;
        length_plotted_graph = (float) 0.0;
        over_x1 = g_x2;

        for( plot_point = most_recent_data;
                more_old_data && length_plotted_graph < length_graph;
                plot_point--){

                plot_point = (MAX_MEASURED + plot_point) % MAX_MEASURED;
                previous_point = (MAX_MEASURED + plot_point - 1) % MAX_MEASURED;
```

```
if (data[previous_point][di] == 999) {
        more_old_data = 0;
        goto end_lp2;
}
old_over_x1 = over_x1;
over_x1 += (float) data[plot_point][di];
length_plotted_graph += (float) data[plot_point][di];

if (data[plot_point][gr] == 1) {
/* Draw portion of ecg graph. */
        if (data[plot_point][gr] != old_g1_graph) {
                setcolor( &bright_green );
                ecg_g1_oldval = (float) data[plot_point][ec];
                if (old_g1_graph == 0){
                        movabs(&g_x2, &ecg_g1_oldval);
                }
                else{
                        movabs(&old_over_x1, &ecg_g1_oldval);
                }
        }
        ecg_g1_oldval = (float) data[previous_point][ec];
/* For last point that will be plotted on graph
        (left_most edge) compute the endpoint of the portion
        of the graph that is being graphed. */
        if (over_x1 <= g_x1) {
         length_left = old_over_x1 - g_x1 - (float) 1.0;
         percent_of_line = length_left / (float) data[plot_point][di];
         y_difference = (float) (data[previous_point][ec]
                                - data[plot_point][ec]);
         ecg_g1_oldval = (float) data[plot_point][ec]
                                + percent_of_line * y_difference;
         over_x1 = g_x1 + (float)1.0;
        }

        lnabs(&over_x1, &ecg_g1_oldval);
}

else if (data[plot_point][gr] == 2
                        || data[plot_point][gr] == 3) {
/* Draw portion of cuff graph. */
        if ( data[plot_point][gr] == 2){
                if (data[plot_point][gr] != old_g1_graph) {
                        setcolor( &bright_blue );
                        cuff_g1_oldval = (float) data[plot_point][cu];
                        if (old_g1_graph == 0){
                        movabs(&g_x2, &cuff_g1_oldval);
                        }
                        else{
                        movabs(&old_over_x1, &cuff_g1_oldval);
                        }
                }
                cuff_g1_oldval = (float) data[previous_point][cu];
/* For last point that will be plotted on graph
         (left_most edge) compute the endpoint of the portion
        of the graph that is being graphed. */
                if (over_x1 <= g_x1) {
                        length_left = old_over_x1 - g_x1
                                                - (float) 1.0;
                        percent_of_line = length_left
                                        / (float) data[plot_point][di];
```

```
                        y_difference = (float) (data[previous_point][cu]
                                        - data[plot_point][cu]);
                        cuff_g1_oldval = (float) data[plot_point][cu]
                                + percent_of_line * y_difference;
                        over_x1 = g_x1 + (float)1.0;
                }

                lnabs(&over_x1  &cuff_g1_oldval);
        }
/* Draw portion of venous graph. */
        if ( data[plot_point][gr] == 3) {
                if (data[plot_point][gr] != old_g1_graph) {
                        setcolor( &bright_pink );
                        venous_g1_oldval = (float) data[plot_point][ve];
                        if (old_g1_graph == 0){
                        movabs(&g_x2, &venous_g1_oldval);
                        }
                        else{
                        movabs(&old_over_x1, &venous_g1_oldval);
                        }
                }
                venous_g1_oldval = (float)data[previous_point][ve];
/* For last point that will be plotted on graph
        (left_most edge) compute the endpoint of the portion
        of the graph that is being graphed. */
                if (over_x1 <= g_x1) {
                        length_left = old_over_x1 - g_x1
                                        - (float) 1.0;
                        percent_of_line = length_left
                                        / (float) data[plot_point][di];
                        y_difference = (float) (data[previous_point][ve]
                                        - data[plot_point][ve]);
                        venous_g1_oldval = (float) data[plot_point][ve]
                                + percent_of_line * y_difference;
                        over_x1 = g_x1 + (float)1.0;
                }

                lnabs(&over_x1, &venous_g1_oldval);
        }
}
else if (data[plot_point][gr] == 4) {
/* Draw portion of pps graph. */
        if (data[plot_point][gr] != old_g1_graph) {
                setcolor( &bright_orange );
                pps_g1_oldval = (float) data[plot_point][pp];
                if (old_g1_graph == 0){
                movabs(&g_x2, &pps_g1_oldval);
                }
                else{
                movabs(&old_over_x1, &pps_g1_oldval);
                }
        }
        pps_g1_oldval = (float) data[previous_point][pp];

/* For last point that will be plotted on graph
        (left_most edge) compute the endpoint of the portion
        of the graph that is being graphed. */
        if (over_x1 <= g_x1) {
         length_left = old_over_x1 - g_x1 - (float) 1.0;
         percent_of_line = length_left/ (float) data[plot_point][di];
         y_difference = (float) (data[previous_point][pp]
```

```
                                        * data[plot_point][pp]);
                        pps_g1_oldval = (float) data[plot_point][pp]
                                        + percent_of_line * y_difference;
                        over_x1 = g_x1 + (float)1.0;
                       }

                        lnabs(&over_x1, &pps_g1_oldval);
                }

        old_g1_graph = data[plot_point][gr];

end_lp2:
                ;
        }

        /* Calculate the number of milliseconds represented over the length of the graph. */
        temp_recent_data = most_recent_data;
        if (temp_recent_data < plot_point) temp_recent_data += MAX_MEASURED;
        num_pts_plotted = temp_recent_data - plot_point;
        time_over_graph = (int) (((double) num_pts_plotted / INT_FREQ) * 1000.0);
        if (time_over_graph != old_msec) {
                setcolor( &blue );
                bar( &msec_x1, &msec_y1, &msec_x2, &msec_y2 );
                if ( !more_old_data ) goto done_msec;
                old_msec = time_over_graph;
                setstext(&msec_height, &aspect, &path);
                setstclr( &light_blue, &light_blue );
                movtcurabs(&msec_x1, &msec_y1);
                itoa(time_over_graph, msec_string, 10);
                stext( msec_string );
                deltcur();
        }
done_msec:
        ;
  }
  /* Draw bp graph when have just frozen or it is time. */
  if (update == GRAPHS_EVERY_X || freeze == 1) {
        more_old_data = 1;
        length_plotted_graph = (float) 0.0;
        over_x1 = g_x2;
        setcolor( &yellow );

        newval = (float) data[most_recent_data][ar];
        movabs(&g_x2, &newval);

        for( plot_point = most_recent_data;
                more_old_data && length_plotted_graph < length_graph;
                plot_point--){

                plot_point = (MAX_MEASURED +     plot_point) % MAX_MEASURED;
                previous_point = (MAX_MEASURED + plot_point - 1) % MAX_MEASURED;

                if (data[previous_point][di] == 999) {
                        more_old_data = 0;
                        goto end_lp3;
                }

                old_over_x1 = over_x1;
                over_x1 -= (float) data[plot_point][di];
                 length_plotted_graph += (float) data[plot_point][di];
```

```
                /* Draw portion of bp graph. */

                g2_oldval = (float) data[previous_point][ar];

                /* For last point that will be plotted on graph
                        (left_most edge) compute the endpoint of the portion
                        of the graph that is being graphed. */
                if (over_x1 <= g_x1) {
                        length_left = old_over_x1 - g_x1 - (float) 1.0;
                        percent_of_line = length_left/ (float) data[plot_point][di];
                        y_difference = (float) (data[previous_point][ar]
                                                - data[plot_point][ar]);
                        g2_oldval = (float) data[plot_point][ar]
                                                + percent_of_line * y_difference;
                        over_x1 = g_x1 + (float)1.0;
                }

                inabs(&over_x1, &g2_oldval);

end_lp3:
                ;
        }
  }
  if (freeze == 1) frozen = 1;
 }
}

void update_process(int pps,int cuff,int venous,int core,int skin)
{
        int color;
        float cursor_x = (float) 165.0, cursor_y = (float) 245.0;
        double angle;
        float center_x = (float) 170.0, center_y = (float) 182.0;
        float small_radius = (float) 4.0;
        float very_small_radius = (float) 2.0;
        int normal_width = 1;
        int odd_number_line_width = 3;
        float length_line = (float) 20.0;
        float move_x, move_y, temporary;
        float twice_x, twice_y;
        float cos_angle, sin_angle;
        float roller_radius = (float) 7.0;
        int i;
        double angle_radians;
        static double current_angle = 360.0;
        int angle_int;
        double pps_constant = (float) 0.20, loop_time = (float) ( 10.0 / INT_FREQ );
        float pump_radius = (float) 28.0;

        int hatch_index = 1;
        int path = 0;
        float height = (float) 40.0;

        float cuff_x = (float) 409.0, cuff_y = (float) 261.0;
        float venous_x = (float) 460.0, venous_y = (float) 135.0;
        float core_x = (float) 700.0, core_y = (float) 135.0;
        float skin_x = (float) 800.0, skin_y = (float) 251.0;
        float plus = (float) 5.0;
        float cuff_yp = cuff_y + plus;
        float venous_yp  = venous_y + plus;
        float core_yp = core_y + plus;
```

```
float skin_yp = skin_y + plus;
float cuff_x3 = (float) 479.0, cuff_y3 = (float) 300.0;
float venous_x3 = (float) 530.0, venous_y3 = (float) 174.0;
float core_x3 = (float) 753.0, core_y3 = (float) 174.0;
float skin_x3 = (float) 843.0, skin_y3 = (float) 292.0;
char string[3];
char just_string[4];
char temp_string[3];
int write_cuff, write_venous, write_core, write_skin;
static int old_cuff_dig = -99;
static int old_venous_dig = -99;
static int old_core_dig = -99;
static int old_skin_dig = -99;

int cuff_line_width;
float top_start_cuff_x = (float) 178.0;
float top_start_cuff_y = (float) 295.0;
float bottom_start_cuff_x = (float) 175.0;
float bottom_start_cuff_y = (float) 91.0;
float top_cuff_x, top_cuff_y, bottom_cuff_x, bottom_cuff_y;
float top_cuff_length_x = (float) 200.0;
float top_cuff_length_y = (float) -24.0;
float bottom_cuff_length_x = (float) 200.0;
float bottom_cuff_length_y = (float) -19.0;
float bottom_line[9][2];
float top_line[9][2];
int num_lines, j, x = 0, y = 1;
static int old_num_lines = 0;

float mercury_height;
float skin_x1 = (float) 792.0, skin_y1;
float skin_x2 = (float) 796.0, skin_y2 = (float) 267.0;
float core_x1 = (float) 692.0, core_y1;
float core_x2 = (float) 696.0, core_y2 = (float) 132.0;
float therm_width = (float) 5.0;
float back_a_bit = (float) -4.0, down_a_bit = (float) -4.0;
float no_change = (float) 0.0;
float max_mercury = (float) 50.0;
float max_cory1 = (float) 185.0;
float max_skny1 = (float) 320.0;
static int old_skin = -99, old_core = -99;
int draw_skin, draw_core;

/* fldsize;

double needle_angle;
float length_needle = (float) 11.0;
float cuff_center_x = (float) 386.0;
float cuff_center_y = (float) 312.0;
float venous_center_x = (float) 437.0;
float venous_center_y = (float) 167.0;
float gage_radius = (float) 11.0;
static int old_cuff = -99;
static int old_venous = -99;

static int old_arm_color = 0;
static int old_loop_color = 0;

if (process_restart) {
        current_angle = 360.0;
        old_cuff_dig = -99;
```

```
        old_venous_dig = -99;
        old_core_dig = -99;
        old_skin_dig = -99;
        old_num_lines = 0;
        old_skin = -99;
        old_core = -99;
        old_cuff = -99;
        old_venous = -99;

        old_arm_color = 0;
        old_loop_color = 0;
        process_restart = 0;
}


/* Draw arm outline.  Maintain value of old color.  If new
        color is the same as old color, do not draw the arm. */
/* Determine new color. */
if ( skin < 20 ) color = navy;
else if ( skin < 30 ) color = light_blue;
else if ( skin < 35 ) color = skin_pink;
else color = red;
/* If color has changed, draw new arm outline. */
if (old_arm_color != color) {
        setcolor(&color);
        draw_arm_outline();
}
old_arm_color = color;

/* Draw cuff. */

/* Calculate width of cuff. */
if (cuff <= 25) num_lines = 1;
else if (cuff <= 50) num_lines = 2;
else if (cuff <= 75) num_lines = 3;
else if (cuff <= 100) num_lines = 4;
else if (cuff <= 125) num_lines = 5;
else if (cuff <= 150) num_lines = 6;
else if (cuff <= 200) num_lines = 7;
else if (cuff <= 250) num_lines = 8;
else num_lines = 9;
if (old_num_lines == num_lines) goto end_draw_cuff;

top_line[0][1] = (float) 0.0;
top_line[1][1] = (float) 3.0;
top_line[2][1] = (float) 6.0;
top_line[3][1] = (float) 9.0;
top_line[4][1] = (float) 12.0;
top_line[5][1] = (float) 15.0;
top_line[6][1] = (float) 17.0;
top_line[7][1] = (float) 20.0;
top_line[8][1] = (float) 23.0;

for (i=0;i<=8;i++) {
        j = 0;
        top_line[i][j] = (float)1.0;
        bottom_line[i][j] = (float)1.0;
        j = 1;
        bottom_line[i][j] = top_line[i][j];
}
```

```
        cuff_line_width = 1;
        setlnwidth( &cuff_line_width );

        if (num_lines < old_num_lines) {
                setcolor( &white );
                for (j=old_num_lines; j>=num_lines+1; j--) {
                        i=j-1;
                        top_cuff_x = top_start_cuff_x + top_line[i][x];
                        top_cuff_y = top_start_cuff_y + top_line[i][y];
                        movabs( &top_cuff_x, &top_cuff_y );
                        lnrel( &top_cuff_length_x, &top_cuff_length_y );
                        bottom_cuff_x = bottom_start_cuff_x - bottom_line[i][x];
                        bottom_cuff_y = bottom_start_cuff_y - bottom_line[i][y];
                        movabs( &bottom_cuff_x, &bottom_cuff_y );
                        lnrel(&bottom_cuff_length_x,&bottom_cuff_length_y);
                }
        }
        if (num_lines > old_num_lines) {
                setcolor( &blue );
                for (j=old_num_lines+1; j<=num_lines; j++) {
                        i=j-1;
                        top_cuff_x = top_start_cuff_x + top_line[i][x];
                        top_cuff_y = top_start_cuff_y + top_line[i][y];
                        movabs( &top_cuff_x, &top_cuff_y );
                        lnrel( &top_cuff_length_x, &top_cuff_length_y );
                        bottom_cuff_x = bottom_start_cuff_x - bottom_line[i][x];
                        bottom_cuff_y = bottom_start_cuff_y - bottom_line[i][y];
                        movabs( &bottom_cuff_x, &bottom_cuff_y );
                        lnrel(&bottom_cuff_length_x,&bottom_cuff_length_y);
                }
        }
        old_num_lines = num_lines;

        setlnwidth( &normal_width );

end_draw_cuff:

        /* Draw mercury in thermometers. */

        /* Calculate mercury height for skin temp.  If hasn't changed,
                set so that won't draw new mercury. */
        mercury_height = ((float) skin - (float) 30.0) / (float)0.20;
        if (mercury_height < (float) 0.0)
                                mercury_height = (float) 0.0;
        if (mercury_height > max_mercury)
                                mercury_height = max_mercury;

        draw_skin = 1;
        if ((int)mercury_height == old_skin) draw_skin = 0;

        /* If new mercury is needed for skin temp, draws the mercury.
                If new mercury height is smaller than old height, first
                draws white bar to erase old mercury.  Then, regardless,
                draws the new mercury. */
        if (draw_skin == 1) {
                if ((int)mercury_height < old_skin) {
                setcolor( &white );
                skin_y1 = skin_y2 + max_mercury + (float) 3.0;
                bar( &skin_x1, &skin_y1, &skin_x2, &skin_y2 );
                }
                setcolor( &light_gray );
```

```
        skin_y1 = skin_y2 + mercury_height + (float) 3.0;
    bar( &skin_x1, &skin_y1, &skin_x2, &skin_y2 );
    }
    old_skin = (int) mercury_height;

    /* Calculate mercury height for core temp.  If hasn't changed,
            set so that won't draw new mercury. */
    mercury_height = ((float) core - (float) 30.0) / (float)0.20;
    if (mercury_height < (float) 0.0)
                                mercury_height = (float) 0.0;
    if (mercury_height > max_mercury)
                                mercury_height = max_mercury;

    draw_core = 1;
    if ((int)mercury_height == old_core) draw_core = 0;

    /* If new mercury is needed for core temp, draws the mercury.
            If new mercury height is smaller than old height, first
            draws white bar to erase old mercury.  Then, regardless,
            draws the new mercury. */
    if (draw_core == 1 || old_core == -99) {
            if ((int)mercury_height < old_core) {
                    setcolor( &white );
                    core_y1 = core_y2 + max_mercury + (float) 3.0;
                    bar( &core_x1, &core_y1, &core_x2, &core_y2 );
            }
            setcolor( &light_gray );
            core_y1 = core_y2 + mercury_height + (float) 3.0;
            bar( &core_x1, &core_y1, &core_x2, &core_y2 );
    }
    old_core = (int) mercury_height;

    /* Draw needles on cuff and venous pressure guages. */

    /* If the cuff pressure has changed, erase old needle,
            calculate new needle position, and draw new needle. */
    if (cuff != old_cuff){
            /* Erase old needle by drawing blue circle. */
            movabs( &cuff_center_x, &cuff_center_y );
            setcolor( &bright_blue );
            fcir( &gage_radius );
            /* Draw pin for needle. */
            setcolor( &black );
            fcir( &very_small_radius );
            /* Calculate end point of new needle. */
            needle_angle = -140.0 * (double) cuff / 300.0 + 160.0;
            angle_int = (int) needle_angle;
            angle_radians = (double) angle_int * PI / 180.0;
                    cos_angle = (float) cos( angle_radians );
            sin_angle = (float) sqrt( 1.0 - (double)(cos_angle*cos_angle) );
            move_x = cos_angle *   ngth_needle;
            move_y = sin_angle * length_needle;
            /* Draw new needle. */
            move_y *= aspect;
            lnrel( &move_x, &move_y );
    }
    old_cuff = cuff;

    /* If the venous pressure has changed, erase old needle,
            calculate new needle position, and draw new needle. */
    if ( venous != old_venous ){
```

```
        /* Erase old needle by drawing blue circle. */
        movabs( &venous_center_x, &venous_center_y );
        setcolor( &bright_pink );
        fcir( &gage_radius );
        /* Draw pin for needle. */
        setcolor( &black );
        fcir( &very_small_radius );
        /* Calculate end point of new needle. */
        needle_angle = -140.0 * (double) venous / 300.0 + 160.0;
        angle_int = (int) needle_angle;
        angle_radians = (double) angle_int * PI / 180.0;
                cos_angle = (float) cos( angle_radians );
        sin_angle = (float) sqrt( 1.0 - (double)(cos_angle*cos_angle) );
        move_x = cos_angle * length_needle;
        move_y = sin_angle * length_needle;
        /* Draw new needle. */
        move_y *= aspect;
        lnrel( &move_x, &move_y );
}
old_venous = venous;

/* Fill in loop outline.  Maintain value of old color.  If
        new color is the same as old color, do not refill the
        loop. */
/* Determine new color. */
if ( core < 20 ) color = navy;
else if ( core < 30 ) color = light_blue;
else if ( core < 35 ) color = skin_pink;
else color = red;
/* If color has changed, refill loop. */
if (color != old_loop_color) {
        movabs( &cursor_x, &cursor_y );
        flood(&color);
}
old_loop_color = color;

/* Draw pump head every time an icu graph is updated. */
if (update == GRAPHS_EVERY_X) {
        current_angle -= (pps_constant * (double) pps * loop_time);
        movabs( &center_x, &center_y );
        setcolor( &white );
        fcir( &pump_radius );
        setcolor( &black );
        fcir( &small_radius );
        current_angle = (double) ((int) current_angle % 90);
        angle = current_angle * PI / 180.0;
        setlnwidth(&odd_number_line_width);
                cos_angle = (float) cos( angle );
        sin_angle = (float) sqrt( 1.0 - (double)(cos_angle*cos_angle) );
        move_x = cos_angle * length_line;
        move_y = sin_angle * length_line;
        /* Move cursor to end of arm in first quadrant. */
        temporary = move_y * aspect;
        movrel( &move_x, &temporary );
        /* Draw arms and rollers in first and third quadrant. */
        fcir( &roller_radius );
        twice_x = (float)-2.0 * move_x;
        twice_y = (float)-2.0 * move_y;
        temporary = aspect * twice_y;
        lnrel( &twice_x, &temporary);
        fcir( &roller_radius );
```

```
            movabs( &center_x, &center_y );

            /* Move cursor to end of arm in second quadrant. */
            temporary = (float)-1.0 * move_y;
            move_y = move_x;
            move_x = temporary;
            temporary = move_y * aspect;
            movrel( &move_x , &temporary );
            /* Draw arms and rollers in second and fourth quadrant. */
            fcir( &roller_radius );
            twice_x = (float)-2.0 * move_x;
            twice_y = (float)-2.0 * move_y;
            temporary = aspect * twice_y;
            lnrel( &twice_x, &temporary);
            fcir( &roller_radius );
    }

    setlnwidth( &normal_width );

    /* Write in numbers that have changed.  For each number that
            is written, first draw a bar to erase the old number,
            then convert the integer value to a string and write the
            string. */
    write_cuff = write_venous = write_core = write_skin = 1;
    if (cuff == old_cuff_dig) write_cuff = 0;
    if (venous == old_venous_dig) write_venous = 0;
    if (core == old_core_dig) write_core = 0;
    if (skin == old_skin_dig) write_skin = 0;
    old_cuff_dig = cuff;
    old_venous_dig = venous;
    old_core_dig = core;
    old_skin_dig = skin;

    setcolor( &white );
    sethatchstyle(&hatch_index);
    setstext(&height, &aspect, &path);
    if (write_cuff) {
            bar( &cuff_x, &cuff_yp, &cuff_x3, &cuff_y3 );
            setstclr(&blue, &blue);
            movtcurabs(&cuff_x, &cuff_y);
            itoa( cuff, string, 10 );
         fldsize = 3;
            if (cuff < 10) fldsize = 2;
            stpjust( just_string, string, ' ', fldsize, JUST_RIGHT);
            stext( just_string );
    }
    if (write_venous) {
            bar( &venous_x, &venous_yp, &venous_x3, &venous_y3 );
            setstclr(&bright_pink, &bright_pink),
            movtcurabs(&venous_x, &venous_y);
            itoa( venous, string, 10 );
         fldsize = 3;
            if (venous < 10) fldsize = 2;
            stpjust( just_string, string, ' ', fldsize, JUST_RIGHT);
            stext( just_string );
    }
    if (write_core) {
            bar( &core_x, &core_yp, &core_x3, &core_y3 );
            setstclr(&red, &red);
            movtcurabs(&core_x, &core_y);
            itoa( core, string, 10 );
```

```
            fldsize = 2;
               if (core < 10) fldsize = 1;
               stpjust( temp_string, string, ' ', fldsize, JUST_RIGHT);
               stext( temp_string );
        }
        if (write_skin) {
               bar( &skin_x, &skin_yp, &skin_x3, &skin_y3 );
               setstclr(&red, &red);
               movecurabs(&skin_x, &skin_y);
               itoa( skin, string, 10 );
            fldsize = 2;
               if (skin < 10) fldsize = 1;
               stpjust( temp_string, string, ' ', fldsize, JUST_RIGHT);
               stext( temp_string );
        }
        deltcur();
}

void draw_arm_outline()
{
        float arm_start_x = (float) 125.0;
        float arm_start_y = (float) 300.0;
        float upper_bicep_x = (float) 310.0;
        float upper_bicep_y = (float) -40.0;
        float upper_forearm_x = (float) 310.0;
        float upper_forearm_y = (float) -3.0;
        float upper_palm_x = (float) 70.0;
        float upper_palm_y = (float) 30.0;

        float upper_thumb_x = (float) 80.0;
        float upper_thumb_y = (float) 10.0;
        float thumb_radius = (float) 9.0;
        float thumb_angle1 = (float) 270.0;
        float thumb_angle2 = (float) 90.0;
        float over_thumb = (float) 0.0;
        float down_thumb = (float) -12.0;
        float who_cares, end_arc_x, end_arc_y;
        float lower_thumb_x = (float) -40.0;
        float lower_thumb_y = (float) -20.0;
        float overlap_x = (float) -25.0, overlap_y = (float) -30.0;

        float bottom_start_y, arm_width = (float) 200.0;
        float lower_bicep_x = (float) 320.0;
        float lower_bicep_y = (float) -30.0;
        float lower_forearm_x = (float) 310.0;
        float lower_forearm_y = (float) 40.0;
        float lower_palm_x = (float) 100.0;
        float lower_palm_y = (float) -20.0;

        float lower_pinky_x = (float) 100.0;
        float lower_pinky_y = (float) 15.0;
        float over_pinky = (float) 1.0, up_pinky = (float) 12.0;
        float pinky_radius = (float) 6.0;
        float pinky_angle1 = (float) 270.0;
        float pinky_angle2 = (float) 90.0;
        float upper_pinky_x = (float) -90.0;
        float upper_pinky_y = (float) 5.0;

        float lower_ring_x = (float) 100.0;
        float lower_ring_y = (float) 10.0;
        float over_finger = (float) 1.0, up_finger = (float) 13.0;
```

```
float finger_radius = (float) 8.0;
float finger_angle1 = (float) 270.0;
float finger_angle2 = (float) 90.0;
float upper_ring_x = (float) -90.0;
float upper_ring_y = (float) 5.0;

float lower_middle_x = (float) 100.0;
float lower_middle_y = (float) 10.0;
float upper_middle_x = (float) -110.0;
float upper_middle_y = (float) 5.0;

float lower_point_x = (float) 100.0;
float lower_point_y = (float) 10.0;
float upper_point_x = (float) -110.0;
float upper_point_y = (float) 5.0;


movabs(&arm_start_x, &arm_start_y);
lnrel(&upper_bicep_x, &upper_bicep_y);
lnrel(&upper_forearm_x, &upper_forearm_y);
lnrel(&upper_palm_x, &upper_palm_y);

lnrel(&upper_thumb_x, &upper_thumb_y);
movrel(&over_thumb, &down_thumb);
arc(&thumb_radius, &thumb_angle1, &thumb_angle2);
inqarc( &end_arc_x, &end_arc_y, &who_cares, &who_cares );
movabs( &end_arc_x, &end_arc_y );
lnrel( &lower_thumb_x, &lower_thumb_y );
lnrel( &overlap_x, &overlap_y );

bottom_start_y = arm_start_y - arm_width;
movabs(&arm_start_x, &bottom_start_y);
lnrel(&lower_bicep_x, &lower_bicep_y);
lnrel(&lower_forearm_x, &lower_forearm_y);
lnrel(&lower_palm_x, &lower_palm_y);

lnrel(&lower_pinky_x, &lower_pinky_y);
movrel(&over_pinky, &up_pinky);
arc(&pinky_radius, &pinky_angle1, &pinky_angle2);
inqarc( &who_cares, &who_cares, &end_arc_x, &end_arc_y );
movabs( &end_arc_x, &end_arc_y );
lnrel( &upper_pinky_x, &upper_pinky_y );

lnrel( &lower_ring_x, &lower_ring_y );
movrel( &over_finger, &up_finger );
arc(&finger_radius, &finger_angle1, &finger_angle2);
inqarc( &who_cares, &who_cares, &end_arc_x, &end_arc_y );
movabs( &end_arc_x, &end_arc_y );
lnrel( &upper_ring_x, &upper_ring_y );

lnrel( &lower_middle_x, &lower_middle_y);
movrel( &over_finger, &up_finger );
arc(&finger_radius, &finger_angle1, &finger_angle2);
inqarc( &who_cares, &who_cares, &end_arc_x, &end_arc_y );
movabs( &end_arc_x, &end_arc_y );
lnrel( &upper_middle_x, &upper_middle_y );

lnrel( &lower_point_x, &lower_point_y);
movrel( &over_finger, &up_finger );
arc(&finger_radius, &finger_angle1, &finger_angle2);
inqarc( &who_cares, &who_cares, &end_arc_x, &end_arc_y );
```

```
        movabs( &end_arc_x, &end_arc_y );
        inrel( &upper_point_x, &upper_point_y );

}

void draw_loop_outline()
{
        float start_loop_x = (float) 164.0;
        float start_loop_y = (float) 252.0;
        float top_outer_length = (float) 600.0;
        float top_outer_y = (float) 0.0;
        float no_change = (float) 0.0, who_cares;
        float outer_down_center_y = (float) -70.0;
        float outer_up_center_y = (float) 70.0;
        float outer_radius = (float) 45.0;
        float outer_angle1 = (float) 270.0;
        float outer_angle2 = (float) 90.0;
        float end_arc_x, end_arc_y;
        float bottom_outer_length = (float) -600.0;
        float bottom_outer_y = (float) 0.0;

        float inner_loop_over = (float) 6.0;
        float loop_width = (float) -25.0;
        float top_inner_length = (float) 590.0;
        float top_inner_y = (float) 0.0;
        float inner_down_center_y = (float) -45.0;
        float inner_up_center_y = (float) 45.0;
        float inner_radius = (float) 30.0;
        float bottom_inner_length = (float) -590.0;
        float bottom_inner_y = (float) 0.0;
        float down_a_bit = (float) -55.0;


        movabs( &start_loop_x, &start_loop_y );
        inrel( &top_outer_length, &top_outer_y );
        movrel( &no_change, &outer_down_center_y );
        arc( &outer_radius, &outer_angle1, &outer_angle2 );
        inqarc( &end_arc_x, &end_arc_y, &who_cares, &who_cares );
        movabs( &end_arc_x, &end_arc_y );
        inrel( &bottom_outer_length, &bottom_outer_y );
        movrel( &no_change, &outer_up_center_y );
        arc( &outer_radius, &outer_angle2, &outer_angle1 );

        movabs( &start_loop_x, &start_loop_y );
        movrel( &inner_loop_over, &loop_width );
        inrel( &top_inner_length, &top_inner_y );
        movrel( &no_change, &inner_down_center_y );
        arc( &inner_radius, &outer_angle1, &outer_angle2 );
        inqarc( &end_arc_x, &end_arc_y, &who_cares, &who_cares );
        movabs( &end_arc_x, &end_arc_y );
        inrel( &bottom_inner_length, &bottom_inner_y );
        movrel( &no_change, &inner_up_center_y );
        arc( &inner_radius, &outer_angle2, &outer_angle1 );

}

void update_header( char *header )
{
        int path = 0;
        float height = (float) 40.0;
        int hatch_index = 1;
```

```
        float x = (float) 0.0, y = (float) 950.0;
        int count = 0;
        float x1 = (float) 0.0, y1 = (float) 1000.0;
        float x2 = (float) 1000.0, y2 = (float) 955.0;

        setcolor(&bright_blue);
        bar(&x1, &y1, &x2, &y2);

        while ( header[count] != '\0' ) ++count;
        x = (float)475.0 - (float) 0.5 * (float)count * (float)20.0;
        sethatchstyle(&hatch_index);
        setstclr(&white, &white);
        setstext(&height, &aspect, &path);
        movtcurabs(&x, &y);
        stext( header );
        deltcur();

        new_header = 0;
}

void help_system(void)
{
        int curs_pos = 7;
        float x1 = (float) 0.0, y1 = (float) 0.0;
        float x2 = (float) 1000.0, y2 = (float) 1000.0;
        float inx1 = (float) 20.0, iny1 = (float) 78.0;
        float inx2 = (float) 980.0, iny2 = (float) 970.0;
        float title_height = (float) 120.0;
        float title_x = (float) 60.0, title_y = (float) 800.0;
        static char title_mssg[] = { " Help System " };
        float func_mssg_height = (float) 40.0;
        float funckeys_x = (float) 55.0, funckeys_y = (float) 720.0;
        static char funckeys_mssg[] = { "Press function keys for option descriptions." };
        float esc_x = (float) 190.0, esc_y = (float) 650.0;
        static char esc_mssg[] = { "Press ESC to exit help system." };
        float box_x1 = (float) 100.0, box_y1 = (float) 125.0;
        float box_x2 = (float) 900.0, box_y2 = (float) 600.0;
        float message_height = (float) 35.0;
        float message_x = (float) 160.0, message_y1 = (float) 530.0;
        float message_y2 = (float) 485.0;
        float message_y3 = (float) 440.0;
        float message_y4 = (float) 395.0;
        float message_y5 = (float) 350.0;
        float message_y6 = (float) 305.0;
        float message_y7 = (float) 260.0;
        float message_y8 = (float) 215.0;
        float message_y9 = (float) 170.0;
        float message_y10 = (float) 125.0;

        static char freeze_mssg1[] = { "Toggles between freezing and unfreezing" };
        static char freeze_mssg2[] = { "the icu graphs.  Control of blood" };
        static char freeze_mssg3[] = { "pressure is not affected." };

        static char range_mssg1[] = { "Controls the y-axis of the arterial" };
        static char range_mssg2[] = { "graph.  Toggles between a range of" };
        static char range_mssg3[] = { "0 - 300 mmHg and a range of just" };
        static char range_mssg4[] = { "below diastolic to just above systolic." };

        static char decrease_mssg1[] = { "Decreases the speed that data points" };
        static char decrease_mssg2[] = { "move across the graph.  As the data" };
        static char decrease_mssg3[] = { "points move more slowly, more of the" };
```

```
        static char decrease_mssg4[] = { "signal can be seen at one time." };

        static char increase_mssg1[] = { "Increases the speed that data points" };
        static char increase_mssg2[] = { "move across the graph.  As the data" };
        static char increase_mssg3[] = { "points move more quickly, less of the" };
        static char increase_mssg4[] = { "signal is visible at one time.  The" };
        static char increase_mssg5[] = { "signal becomes spread out across the" };
        static char increase_mssg6[] = { "screen." };

        static char settings_mssg1[] = { "Indicates current settings of the header," };
        static char settings_mssg2[] = { "heart rate, systolic, and diastolic " };
        static char settings_mssg3[] = { "values.  Gives the option to restart the" };
        static char settings_mssg4[] = { "simulation with new values.  While the" };
        static char settings_mssg5[] = { "current settings are displayed, the" };
        static char settings_mssg6[] = { "simulation screen disappears from the" };
        static char settings_mssg7[] = { "screen.  However, computer-control of" };
        static char settings_mssg8[] = { "the arterial pressure and measurement" };
        static char settings_mssg9[] = { "of data continue." };


        static char display_mssg1[] = { "Toggles between signals displayed on" };
        static char display_mssg2[] = { "the upper icu graph." };
        static char display_mssg3[] = { "   ECG          -- green " };
        static char display_mssg4[] = { "   Cuff Pressure   -- blue" };
        static char display_mssg5[] = { "   Venous Pressure -- pink" };
        static char display_mssg6[] = { "   Motor Speed     -- orange" };

        static char help_mssg[] = { "Explanation of eight function keys." };

        static char quit_mssg1[] = { "Stops the simulation and exits the" };
        static char quit_mssg2[] = { "program." };

        int pass = 0;
        int ch;

        delbox();
        setscreen(&page2);
        setcolor( &blue );
        bar(&x1, &y1, &x2, &y2);
        setcolor( &white );
        bar(&inx1, &iny1, &inx2, &iny2);
        setstext( &title_height, &aspect, &path );
        setstclr( &blue, &blue );
        movtcurabs( &title_x, &title_y );
        stext( title_mssg );
        setstext( &func_mssg_height, &aspect, &path );
        movtcurabs( &funckeys_x, &funckeys_y );
        stext( funckeys_mssg );
        movtcurabs( &esc_x, &esc_y );
        stext( esc_mssg );
        set_up_options_bar();
        update_options(curs_pos);
        setcolor( &blue );
        bar( &box_x1, &box_y1, &box_x2, &box_y2 );
        setstclr( &white, &white );
        display( &page2 );

keypress:       while ( !kbhit() );
        ch = getch();
        if (ch == '0') ch = getch();
```

```
if (ch == 27) {
        delbox();
        setscreen( &page1 );
        display( &page1 );
        return;
}
else
if (ch >= 59 && ch <= 66) curs_pos = ch - 58;
else
if (ch >= 49 && ch <= 56) curs_pos = ch - 48;
else
if (ch == 72 || ch == 75) curs_pos--;
else
if (ch == 77 || ch == 80) curs_pos++;
else
if (ch == 13);
else goto keypress;

if (curs_pos >= 9) curs_pos = 1;
else
if (curs_pos <= 0) curs_pos = 8;

update_options(curs_pos);
setcolor( &blue );
bar( &box_x1, &box_y1, &box_x2, &box_y2 );
setstext( &message_height, &aspect, "path" );
movtcurabs( &message_x, &message_y1 );

switch(curs_pos)
{
        case 1:
                stext( freeze_mssg1 );
                movtcurabs( &message_x, &message_y2 );
                stext( freeze_mssg2 );
                movtcurabs( &message_x, &message_y3 );
                stext( freeze_mssg3 );
                break;
        case 2:
                stext( range_mssg1 );
                movtcurabs( &message_x, &message_y2 );
                stext( range_mssg2 );
                movtcurabs( &message_x, &message_y3 );
                stext( range_mssg3 );
                movtcurabs( &message_x, &message_y4 );
                stext( range_mssg4 );
                break;
        case 3:
                stext( decrease_mssg1 );
                movtcurabs( &message_x, &message_y2 );
                stext( decrease_mssg2 );
                movtcurabs( &message_x, &message_y3 );
                stext( decrease_mssg3 );
                movtcurabs( &message_x, &message_y4 );
                stext( decrease_mssg4 );
                break;
        case 4:
                stext( increase_mssg1 );
                movtcurabs( &message_x, &message_y2 );
                stext( increase_mssg2 );
                movtcurabs( &message_x, &message_y3 );
                stext( increase_mssg3 );
```

```
                        movtcurabs( &message_x, &message_y4 );
                        stext( increase_mssg4 );
                        movtcurabs( &message_x, &message_y5 );
                        stext( increase_mssg5 );
                        movtcurabs( &message_x, &message_y6 );
                        stext( increase_mssg6 );
                        break;
                case 5:
                        stext( settings_mssg1 );
                        movtcurabs( &message_x, &message_y2 );
                        stext( settings_mssg2 );
                        movtcurabs( &message_x, &message_y3 );
                        stext( settings_mssg3 );
                        movtcurabs( &message_x, &message_y4 );
                        stext( settings_mssg4 );
                        movtcurabs( &message_x, &message_y5 );
                        stext( settings_mssg5 );
                        movtcurabs( &message_x, &message_y6 );
                        stext( settings_mssg6 );
                        movtcurabs( &message_x, &message_y7 );
                        stext( settings_mssg7 );
                        movtcurabs( &message_x, &message_y8 );
                        stext( settings_mssg8 );
                        movtcurabs( &message_x, &message_y9 );
                        stext( settings_mssg9 );
                        break;
                case 6:
                        stext( display_mssg1 );
                        movtcurabs( &message_x, &message_y2 );
                        stext( display_mssg2 );
                        setstclr( &bright_green, &bright_green );
                        movtcurabs( &message_x, &message_y4 );
                        stext( display_mssg3 );
                        setstclr( &bright_blue, &bright_blue );
                        movtcurabs( &message_x, &message_y5 );
                        stext( display_mssg4 );
                        setstclr( &bright_pink, &bright_pink );
                        movtcurabs( &message_x, &message_y6 );
                        stext( display_mssg5 );
                        setstclr( &bright_orange, &bright_orange );
                        movtcurabs( &message_x, &message_y7 );
                        stext( display_mssg6 );
                        setstclr( &white, &white );
                        break;
                case 7:
                        stext( help_mssg );
                        break;
                case 8:
                        stext( quit_mssg1 );
                        movtcurabs( &message_x, &message_y2 );
                        stext( quit_mssg2 );
                        break;
        }

        deltcur();
        goto keypress;
}

void update_options(curs_pos)
{
        float y1 = (float) 1.0, y2 = (float) 47.0;
```

```
        static float x1[9] = { (float) 1.0, (float) 88.0,
                (float) 224.0,
                (float) 359.0, (float) 488.0, (float) 601.0,
                (float) 774.0, (float) 840.0 , (float) 997.0 };
        int curs_pos_plus;

        setcolor( &bright_blue );
        curs_pos_plus = curs_pos;
        curs_pos--;
        rbox(&x1[curs_pos], &y1, &x1[curs_pos_plus], &y2);
}

void gms_logo()
{
        float out_x1 = (float) 0.0, out_y1 = (float) 1000.0;
        float out_x2 = (float) 1000.0, out_y2 = (float) 0.0;
        float in_x1 = (float) 20.0, in_y1 = (float) 975.0;
        float in_x2 = (float) 975.0, in_y2 = (float) 25.0;

        int path = 0;
        float text_height = (float) 40.0, copy_height = (float) 25.0;
        float eng_height = (float) 70.0, gms_height = (float) 300.0;
        float gms_x = (float) 200.0, gms_y = (float) 600.0;
        static char gms[] = { "GMS" };
        float eng_x = (float) 300.0, eng_y = (float) 575.0;
        static char eng[] = { "ENGINEERING" };
        float corp_x = (float) 300.0, corp_y = (float) 500.0;
        static char corp[] = { "CORPORATION" };
        float address_x = (float) 360.0, address_y = (float) 400.0;
        static char address[] = { "Columbia, MD" };
        float phone_x = (float) 350.0, phone_y = (float) 350.0;
        static char phone[] = { "(301) 995-0508" };
        float copy_x = (float) 450.0, copy_y = (float) 30.0;
        static char copy[] =
                                {"Copyright (c) GMS Engineering Corp 1988"};
        int hatch_index = 1;

        setcolor(&gms_blue);
        bar(&out_x1, &out_y1, &out_x2, &out_y2);
        setcolor(&light_blue);
        bar(&in_x1, &in_y1, &in_x2, &in_y2);
        sethatchstyle(&hatch_index);
        setstclr(&gms_blue, &gms_blue);
        setstext(&gms_height, &aspect, &path);
        movtcurabs(&gms_x, &gms_y);
        stext( gms );
        setstext(&eng_height, &aspect, &path);
        movtcurabs(&eng_x, &eng_y);
        stext( eng );
        movtcurabs(&corp_x, &corp_y);
        stext( corp );
        setstext(&text_height, &aspect, &path);
        movtcurabs(&address_x, &address_y);
        stext( address );
        movtcurabs(&phone_x, &phone_y);
        stext( phone );
        setstext(&copy_height, &aspect, &path);
        movtcurabs(&copy_x, &copy_y);
        stext( copy );
        deltcur();
}
```

```
void draw_graph_labels(int g1_graph, int bp_min, int bp_max)
{
        float g1_range, bp_range, g2_range;
        float y_level, top_y_level;
        float ecg_range;
        int ecg_incrmnt = 5;
        int ecg_max = 10, ecg_min = -10;
        float pps_range;
        int pps_incrmnt = 2000;
        int pps_max = 6000, pps_min = 0;
        float press_range;
        int press_incrmnt = 100;
        int press_max = 300, press_min = 0;
        float hatch_length = (float) 3.0;
        float no_change = (float) 0.0;
        float g_x1 = (float) 112.0;
        int g1_y1 = 669;
        float g_x2 = (float) 651.0;
        int g1_y2 = 938;
        float g_x3 = (float) 695.0;
        int g2_y1 = 362, g2_y2 = 631;
        int error = 0;
        static char label[3];
        int path = 0;
        float height = (float) 20.0;
        int hatch_index = 1, i;
        static int old_graph = -1;
        static int old_bp_min = -99, old_bp_max = -99;

        float lbl_x1 = (float) 700.0;
        float lbl_y1 = (float) 655.0;
        float lbl_x2 = (float) 765.0;
        float lbl_y2 = (float) 690.0;
        static char press_lbl[] = { "mmHg" };
        static char pps_lbl[] = { "rpm" };
        static char ecg_lbl[] = { "mV" };

        if (drw_grph_restart) {
                old_graph = -1;
                old_bp_min = -99;
                old_bp_max = -99;
                drw_grph_restart = 0;
        }

        setstclr( &light_blue, &light_blue );
        sethatchstyle(&hatch_index);
        setstext(&height, &aspect, &path,

        if (g1_graph != old_graph){

                g1_range = (float) (g1_y2 - g1_y1);

                setcolor( &blue );
                y_level = (float) (g1_y1 - 15);
                top_y_level = (float) (g1_y2 + 10);
                bar(&g_x2, &y_level, &g_x3, &top_y_level);
                bar(&lbl_x1, &lbl_y1, &lbl_x2, &lbl_y2);

                if (g1_graph == 1) {
                        ecg_range = (float) (ecg_max - ecg_min);
```

```
                i = ecg_min;
                for ( y_level = (float) (g1_y1 - 15);
                        y_level <= (float) g1_y2;
                        y_level += (float)ecg_increment * g1_range / ecg_range ) {
                        itoa(i, label, 10);
                        movtcurabs(&g_x2, &y_level);
                        stext( label );
                        i += ecg_increment;
                }
                movtcurabs(&lbl_x1, &lbl_y1);
                setstclr( &bright_green, &bright_green);
                stext( ecg_lbl );
                setstclr( &light_blue, &light_blue );
                deltcur();
        }
        else if (g1_graph == 2 || g1_graph == 3) {
                press_range = (float) (press_max - press_min);
                i = press_min;
                for ( y_level = (float) (g1_y1 - 15);
                   y_level <= (float) g1_y2;
                   y_level += (float)press_increment*g1_range/press_range ){
                        itoa(i, label, 10);
                        movtcurabs(&g_x2, &y_level);
                        stext( label );
                        i += press_increment;
                }
                movtcurabs(&lbl_x1, &lbl_y1);
                if ( g1_graph == 2 ) setstclr( &bright_blue, &bright_blue);
                if ( g1_graph == 3 ) setstclr( &bright_pink, &bright_pink);
                stext( press_lbl );
                setstclr( &light_blue, &light_blue );
                deltcur();
}
        else if (g1_graph == 4) {
                pps_range = (float) (pps_max - pps_min);
                i = pps_min;
                for ( y_level = (float) (g1_y1 - 15);
                        y_level <= (float) g1_y2;
                        y_level += (float)pps_increment * g1_range / pps_range ){
                        itoa(i, label, 10);
                        movtcurabs(&g_x2, &y_level);
                        stext( label );
                        i += pps_increment;
                }
                movtcurabs(&lbl_x1, &lbl_y1);
                setstclr( &bright_orange, &bright_orange );
                stext( pps_lbl );
                setstclr( &light_blue, &light_blue );
                deltcur();
}
        else error = 1;
}
old_graph = g1_graph;

if (bp_min != old_bp_min || bp_max != old_bp_max)
{
        setcolor( &blue );
        y_level = (float) (g2_y1 - 5);
        top_y_level = (float) (g2_y2 + 20);
        bar(&g_x2, &y_level, &g_x3, &top_y_level);
```

```
        if (bp_min == 0) {
              g2_range = (float) (g2_y2 - g2_y1);
              bp_range = (float) (bp_max - bp_min);
              i = 0;
              for ( y_level = (float) (g2_y1 - 5);
                    y_level <= (float) g2_y2;
                    y_level += (float)100 * g2_range / bp_range ){
                    itoa(i, label, 10);
                    movtcurabs(&g_x2, &y_level);
                    stext( label );
                    i += 100;
              }
        }
        else {
              itoa(bp_min, label, 10);
              movtcurabs(&g_x2, &y_level);
              stext( label );
              y_level = top_y_level - (float) 25.0;
              itoa(bp_max, label, 10);
              movtcurabs(&g_x2, &y_level);
              stext( label );
        }
        deltcur();
        }
        old_bp_min = bp_min;
        old_bp_max = bp_max;
}

void contain_vs_settings()
{
        int ch;
        int old_hr_set, old_sys_set;
        int i, digits, max_digits;
        int key, num_scrolled;
        int path = 0, hatch_index = 1;
        float height = (float) 40.0;
        float hr_x = (float) 360.0;
        float sys_x = (float) 535.0;
        float dias_x = (float) 640.0;
        float man_lbl_y = (float) 340.0;
        char digit[4];
        float tcurs_x_pos[3];
        float x_pos;

        int hr_min = 30, hr_max = 150;
        int sys_min = 30, sys_max = 230;
        int dias_min = 10, dias_max = 210;

        float h_x1 = (float) 185.0, h_x2 = (float) 725.0;
        float h_y1 = (float) 558.0, h_y2 = (float) 600.0;
        float m_x1 = (float) 125.0, m_x2 = (float) 290.0;
        float m_y1 = (float) 340.0, m_y2 = (float) 390.0;
        float m1_x1 = (float) 355.0, m1_x2 = (float) 425.0;
        float m1_y1 = (float) 335.0, m1_y2 = (float) 390.0;
        float m2_x1 = (float) 530.0, m2_x2 = (float) 605.0;
        float m2_y1 = (float) 335.0, m2_y2 = (float) 390.0;
        float m3_x1 = (float) 635.0, m3_x2 = (float) 715.0;
        float m3_y1 = (float) 335.0, m3_y2 = (float) 390.0;
        float a1_x1 = (float) 205.0, a1_x2 = (float) 740.0;
        float a1_y1 = (float) 285.0, a1_y2 = (float) 335.0;
        float a2_y1 = (float) 230.0, a2_y2 = (float) 280.0;
```

```
        float a3_y1 = (float) 175.0, a3_y2 = (float) 225.0;
        float a4_y1 = (float) 120.0, a4_y2 = (float) 170.0;
        float a5_y1 = (float)  65.0, a5_y2 = (float) 115.0;

        static int auto_hr[4] = {60,80,72,40};
        static int auto_sys[4] = {120,90,160,150};
        static int auto_dias[4] = {80,50,90,70};

        char string[3];

        old_hr_set = hr_set;
        old_sys_set = sys_set;

header: setcolor( &yellow );
        rbox(&h_x1, &h_y1, &h_x2, &h_y2);
        scpage(page1);
        scapage(page1);
        sccurset(10,16);
        kbquery(header, sizeof(header) , &key, &num_scrolled);
        stpcvt(header, TOUP);

        setstclr(&white, &white);
        sethatchstyle(&hatch_index);
        setstext(&height, &aspect, &path);
manual: rbox(&m_x1, &m_y1, &m_x2, &m_y2);
        while ( !kbhit() );
        ch = getch();
        if (ch == 27) {
                exit_flag = 11;
                goto end_rout;
        }
        if (ch == 0) ch = getch();
        if (ch == 49 || ch == 59) goto auto_1;
        if (ch == 50 || ch == 60) goto auto_2;
        if (ch == 51 || ch == 61) goto auto_3;
        if (ch == 52 || ch == 62) goto auto_4;
        if (ch == 72 || ch == 75 || ch == 53 || ch == 63) goto auto_5;
        if (ch == 'h'|| ch == 'H') goto header;
        if (ch == 13 || ch == 'm' || ch == 'M') {
          /* Get hr. */
hr:             for (i=0; i<=2; i++){
                        string[i] = ' ';
                }
                rbox(&m1_x1, &m1_y1, &m1_x2, &m1_y2);
                /* Get first digit.  */
                digits = 0;
                while ((ch < 49 || ch > 57) && ch != 27) ch = getch();
                if (ch == 27) goto manual;
                movtcurabs(&hr_x, &man_lbl_y);
                itoa(ch-48, digit, 10);
                stext( digit );
                deltcur();
                string[digits] = digit[0];
                ch = 1;
                digits++;
                max_digits = 3;
                /* Get remaining digits and convert to integer form. */
                tcurs_x_pos[0] = hr_x;
                digits = read_value(string, digits, max_digits, tcurs_x_pos);
                if (digits == 99) {             /* Escape has been pushed. */
                        setcolor( &blue );
```

```
                        delbox();
                        bar(&m1_x1, &m1_y1, &m1_x2, &m1_y2);
                        setcolor( &yellow );
                        goto manual;
                }
                hr_set = atoi(string);

                if ( hr_set < hr_min || hr_set > hr_max) {
                        utsound(600,9);
                        setstclr(&blue, &blue);
                        for ( digits = 0; digits <= max_digits; digits++){
                                x_pos = tcurs_x_pos[digits];
                                movtcurabs(&x_pos, &man_lbl_y);
                                digit[0] = string[digits];
                                stext( digit );
                                string[digits] = ' ';
                        }
                        deltcur();
                        setstclr(&white, &white);
                        goto hr;
                }

        /* Get sys. */
sys:            for (i=0; i<=2; i++){
                        string[i] = ' ';
                }
                rbox(&m2_x1, &m2_y1, &m2_x2, &m2_y2);
                /* Get first digit.  */
                digits = 0;
                while ((ch < 49 || ch > 57) && ch != 27) ch = getch();
                if (ch == 27) {
                        hr_set = old_hr_set;
                        goto manual;
                }
                movtcurabs(&sys_x, &man_lbl_y);
                itoa(ch-48, digit, 10);
                stext( digit );
                deltcur();
                string[digits] = digit[0];
                ch = 1;
                digits++;
                max_digits = 3;
                /* Get remaining digits and convert to integer form. */
                tcurs_x_pos[0] = sys_x;
                digits = read_value(string, digits, max_digits, tcurs_x_pos);
                if (digits == 99) {             /* Escape has been pushed. */
                        setcolor( &blue );
                        delbox();
                        bar(&m1_x1, &m1_y1, &m1_x2, &m1_y2);
                        bar(&m2_x1, &m2_y1, &m2_x2, &m2_y2);
                        setcolor( &yellow );
                        hr_set = old_hr_set;
                        goto manual;
                }
                sys_set = atoi(string);

                if ( sys_set < sys_min || sys_set > sys_max ) {
                        utsound(600, 9);
                        setstclr(&blue, &blue);
                        for ( digits = 0; digits <= max_digits; digits++){
                                x_pos = tcurs_x_pos[digits];
```

```
                        movtcurabs(&x_pos, &man_lbl_y);
                        digit[0] = string[digits];
                        stext( digit );
                        string[digits] = ' ';
                }
                deltcur();
                setstclr(&white, &white);
                goto sys;
        }

    /* Get dias. */
dias.   for (i=0; i<=2; i++){
                string[i] = ' ';
        }
        rbox(&m3_x1, &m3_y1, &m3_x2, &m3_y2);
        /* Get first digit.  */
        digits = 0;
        while ((ch < 49 || ch > 57) && ch != 27) ch = getch();
        if (ch == 27) {
                hr_set = old_hr_set;
                sys_set = old_sys_set;
                goto manual;
        }
        movtcurabs(&dias_x, &man_lbl_y);
        itoa(ch-48, digit, 10);
        stext( digit );
        deltcur();
        string[digits] = digit[0];
        ch = 1;
        digits++;
        max_digits = 3;
        /* Get remaining digits and convert to integer form. */
        tcurs_x_pos[0] = dias_x;
        digits = read_value(string, digits, max_digits, tcurs_x_pos);
        if (digits == 99) {             /* Escape has been pushed. */
                setcolor( &blue );
                delbox();
                bar(&m1_x1, &m1_y1, &m1_x2, &m1_y2);
                bar(&m2_x1, &m2_y1, &m2_x2, &m2_y2);
                bar(&m3_x1, &m3_y1, &m3_x2, &m3_y2);
                setcolor( &yellow );
                hr_set = old_hr_set;
                sys_set = old_sys_set;
                goto manual;
        }
        dias_set = atoi(string);

        if ( dias_set < dias_min || dias_set >= sys_set) {
                utsound(600, 9);
                setstclr(&blue, &blue);
                for ( digits = 0; digits <= max_digits; digits++){
                        x_pos = tcurs_x_pos[digits];
                        movtcurabs(&x_pos, &man_lbl_y);
                        digit[0] = string[digits];
                        stext( digit );
                        string[digits] = ' ';
                }
                deltcur();
                setstclr(&white, &white);
                goto dias;
        }
```

```
        }
        else if (ch == 'a' || ch == 'A' || ch == 77 || ch == 80) {
auto_1: rbox(&a1_x1, &a1_y1, &a1_x2, &a1_y2);
                while ( !kbhit() );
                ch = getch();
                if (ch == 27) {
                        exit_flag = 11;
                        goto end_rout;
                }
                if (ch == 0) ch = getch();
                if (ch == 49 || ch == 59) goto auto_1;
                if (ch == 51 || ch == 61) goto auto_3;
                if (ch == 52 || ch == 62) goto auto_4;
                if (ch == 53 || ch == 63) goto auto_5;
                if (ch == 'h' || ch == 'H') goto header;
                if (ch == 72 || ch == 75 || ch == 'm'
                        || ch == 'M') goto manual;
                if (ch == 'a' || ch == 'A' || ch == 77
                        || ch == 80 || ch == 50 || ch == 60) {
auto_2:          rbox(&a1_x1, &a2_y1, &a1_x2, &a2_y2);
                 while ( !kbhit() );
                 ch = getch();
                 if (ch == 27) {
                        exit_flag = 11;
                        goto end_rout;
                 }
                 if (ch == 0) ch = getch();
                 if (ch == 72 || ch == 75 || ch == 49 || ch == 59) goto auto_1;
                 if (ch == 50 || ch == 60) goto auto_2;
                 if (ch == 52 || ch == 62) goto auto_4;
                 if (ch == 53 || ch == 63) goto auto_5;
                 if (ch == 'm' || ch == 'M') goto manual;
                 if (ch == 'h' || ch == 'H') goto header;
                 if (ch == 'a' || ch == 'A' || ch == 77 || ch == 80
                        || ch == 51 || ch == 61) {
auto_3:           rbox(&a1_x1, &a3_y1, &a1_x2, &a3_y2);
                  while ( !kbhit() );
                  ch = getch();
                  if (ch == 27) {
                        exit_flag = 11;
                        goto end_rout;
                  }
                  if (ch == 0) ch = getch();
                  if (ch == 49 || ch == 59) goto auto_1;
                  if (ch == 51 || ch == 61) goto auto_3;
                  if (ch == 53 || ch == 63) goto auto_5;
                  if (ch == 'm' || ch == 'M') goto manual;
                  if (ch == 'h' || ch == 'H') goto header;
                  if (ch == 72 || ch == 75 || ch == 50 || ch == 60) goto auto_2;
                  if (ch == 'a' || ch == 'A' || ch == 77
                        || ch == 80 || ch == 52 || ch == 62) {
auto_4:            rbox(&a1_x1, &a4_y1, &a1_x2, &a4_y2);
                   while ( !kbhit() );
                   ch = getch();
                   if (ch == 27) {
                        exit_flag = 11;
                        goto end_rout;
                   }
                   if (ch == 0) ch = getch();
                   if (ch == 49 || ch == 59) goto auto_1;
```

```
                    if (ch == 50 || ch == 60) goto auto_2;
                    if (ch == 52 || ch == 62) goto auto_4;
                    if (ch == 'm' || ch == 'M') goto manual;
                    if (ch == 'h' || ch == 'H') goto header;
                    if (ch == 72 || ch == 75 || ch == 51 || ch == 61) goto auto_3;
                    if (ch == 'a' || ch == 'A' || ch == 77
                        || ch == 80 || ch == 53 || ch == 63) {
auto_5:              rbox(&a1_x1, &a5_y1, &a1_x2, &a5_y2);
                     while ( !kbhit() );
                     ch = getch();
                     if (ch == 27) {
                         exit_flag = 11;
                         goto end_rout;
                     }
                     if (ch == 0) ch = getch();
                     if (ch == 50 || ch == 60) goto auto_2;
                     if (ch == 51 || ch == 61) goto auto_3;
                     if (ch == 53 || ch == 63) goto auto_5;
                     if (ch == 'h' || ch == 'H') goto header;
                     if (ch == 72 || ch == 75 || ch == 52 || ch == 62) goto auto_4;
                     if (ch == 'a' || ch == 'A' || ch == 49 || ch == 59) goto auto_1;
                     if (ch == 77 || ch == 80 || ch == 'm' || ch == 'M') goto manual;
                     else {
                         calflag = 11;
                     }
                    }
                    else {
                         hr_set = auto_hr[3];
                         sys_set = auto_sys[3];
                         dias_set = auto_dias[3];
                    }
                   }
                   else {
                    hr_set = auto_hr[2];
                    sys_set = auto_sys[2];
                    dias_set = auto_dias[2];
                   }
                  }
                  else {
                   hr_set = auto_hr[1];
                   sys_set = auto_sys[1];
                   dias_set = auto_dias[1];
                  }
                 }
                 else {
                  hr_set = auto_hr[0];
                  sys_set = auto_sys[0];
                  dias_set = auto_dias[0];
                 }
        }
        else goto manual;
end_rout:
        delbox();
}

int read_value(char *string, int digits, int max_digits, float *tcurs_x_pos)
{
int ch = 0;
char digit[1];
float tcurs_y_pos = (float) 340.0;
float x_pos;
```

```
int index;

/*  Until Enter (13) is pressed, wait for keypresses.  While
        there are less than max_digits, accept new digits (48-57) and
        increment size of string.  When backspace (8) is pressed,
        decrement number of digits.  */

        while (ch != 13 && ch != 27) {
                while ( (ch < 48 || ch > 57) && ch != 13 && ch != 8
                        && ch != 27) ch = getch();
                if (ch == 8) {
                        digits--;
                        x_pos = tcurs_x_pos[digits];
                        movtcurabs(&x_pos, &tcurs_y_pos);
                        setstclr(&blue, &blue);
                        digit[0] = string[digits];
                        stext( digit );
                        deltcur();
                        string[digits] = ' ';
                        movtcurabs(&x_pos, &tcurs_y_pos);
                        setstclr(&white, &white);
                }
                else
                if ( ch >= 48 && ch <= 57) {
                        if (digits == max_digits) utsound(low_pitch, short_duration);
                        else {
                                inqtcur(&x_pos, &tcurs_y_pos, &index);
                                tcurs_x_pos[digits] = x_pos;
                                itoa(ch-48, digit, 10);
                                stext( digit );
                                deltcur();
                                string[digits] = digit[0];
                                digits++;
                        }
                }
                if ( ch == 27 ) digits = 99;
                if ( ch != 13 && ch != 27) ch = 1;
        }
        return (digits);
}

/*                                                                      */
/*              disable_ints( state )                                   */
/*                                                                      */
/* This routine disables the interrupt capability on the DAS-8 board by */
/* writing a 0 to the bit of the control register which is connected to */
/* the output enable line of the tri-state buffer which buffers the out- */
/* put of the interrupt flip-flop.  The rest of the control word is left */
/* unchanged.    The function returns the new control reg contents. */

int disable_ints( state )
int state;
{
   state = state & -ENABLE_INTS ;               /* change only the int. enable bit

*/
   outp( DAS8_CONTROL, state );                 /* write to DAS-8 control register

*/
   return( state );                             /* return the revised control word
*/
```

```
}

/*    This routine is the inverse of the above .                    */
/*    This routine also clears the interrupt ff on the DAS-8 board.          */

int enable_ints( state )
int state;
{
   state = state | ENABLE_INTS ;              /* change only the int. enable bit

*/
   outp( DAS8_CONTROL, state );               /* write to DAS-8 control register
   */
   return( state );                           /* return the revised control word
*/
}

int set_imr( int_chan, on_off )               /* programs the 8259 mask register

*/
int int_chan;                                 /* the interrupt bit to set
*/
int on_off;                                   /*  the value ( 1 or 0 ) to set the mask to
*/
{
   int mask_i, mask_o;
   int i=0;

   mask_i = inp( PIC + 1 );
   if( on_off == ON ){
      mask_o = mask_i & ~( 0x01 << int_chan );}
   else
      mask_o = mask_i | ( 0x01 << int_chan );
   outp( PIC + 1, mask_o );
   while( (mask_i = inp( PIC + 1 ) != mask_o )) /* if mask change didn't
take*/
   {
      outp( PIC, 0x20 );   /*  put out general    EOI instruction      */
      outp( PIC + 1, mask_o);  /*  re-write mask                      */
      if( ++i > 10 )
      {
         printf("failed writing mask\n");
         return( 1 );
      }
   }
   return(0);                                 /* success  */
}

/*                                                                      */
/*                int set_timer( cntr, mode, f_in, f_or_n )             */
/*                                                                      */
/*   This routine initializes the specified counter in an 8254 to the*/
/*   specified mode and waveform period or number of events.          */
/*   BCD mode NOT supported.                                          */
/*   The function returns the 16-bit count written to the 8254 if the*/
/*   requested action was reasonable or possible. A -1 return value  */
/*   implies errant request.                                          */

int set_timer( cntr, mode, f_in, f_or_n )
int cntr;                                     /*   the selected counter          */
int mode;                                     /*   the selected mode             */
```

```
double f_in;                                    /*   the input frequency              */
double f_or_n;                                  /*   output frequency or number of
events */
{
int command;
int count;
long int big_one;

   if((cntr >= 0)&&(cntr < 3)&&(mode >= 0)&&(mode < 6)&&(f_or_n > 0.0))
   {
      command = cntr << 6;                      /* shift counter # to bits 7 and 6  */
      command = command | 0x30;          /* always write a two-byte count    */
      command = command | ( mode << 1 );  /* put mode in bits 3-1       */
      outp( CNT_CTRL , command );         /* write the control word out
*/
      switch ( mode )
      {
        case 0: count = (int) f_or_n - 1;         /* terminal count */
                if( count < 0 )                   /* if count too lo*/
                   return( -1 );
                outp( CNT_0 + cntr, count );              /* lo byte     */
                outp( CNT_0 + cntr, count >> 8 );/* hi byte     */
                break;
        case 1: printf(" Sorry! Mode 1 of the 8254 not currently supported.\n");
                break;
        case 2:
        case 3: big_one = (long)( (f_in/f_or_n ) + .5 );
                if( big_one > 0xFFFF )            /* if count is too big */
                   return( -1 );          /*  return error code  */
                count = (int) big_one;            /* coerce big_one to int*/
                if( count < 2 )           /* if count is too small */
                   return(-1);                    /* return error code  */
                outp( CNT_0 + cntr, 0x00FF & count );         /*  lo byte  */
                outp( CNT_0 + cntr, 0x00FF & (count >> 8)); /*    hi byte  */
                break;
        case 4: printf(" Sorry! Mode 4 of the 8254 not currently supported.\n");
                break;
        case 5: printf(" Sorry! Mode 5 of the 8254 not currently supported.\n");
                break;
       default: return( -1 );                      /*  unknown mode         */
      }
   }
   else                                           /*  error  !  */
      return( -1 );                               /*  errant request   */
   return( 0 );
}

/*                                                                              */
/*                  int read_status( cntr )                                     */
/*                                                                          */
/*   This routine reads the status from the designated counter in the       */
/*   DAS-8's 8254.  The return value is equal to status if the counter   */
/*   number is valid (i.e.  0 <= n <= 2), or -1 if invalid.                 */

int read_status( cntr )
int cntr;
{
   int command;

   if( ( cntr > 0 )&&( cntr < 3 ) )               /* cntr is valid...
*/
```

```
    {
      command = ( 0x02 << cntr ) | 0xE0;  /* latch status only */
      outp( CNT_CTRL, command );          /* write the latch command*/
      return ( inp( CNT_0 + cntr ) );             /* return the status      */
    }
    return( -1 );
}


/*                                                                    */
/*                void arm_isr(pregs, pisrblk, pmsg)                  */
/*                                                                    */
/*                                                                    */
void arm_isr(pregs, pisrblk, pmsg)
ALLREG  *pregs;                         /* all CPU registers when ISR invoked  */
ISRCTRL *pisrblk;                               /* ptr to ISR control block            */
ISRMSG *pmsg;                                   /* used by dispatcher                  */
{
    int dummy;                                  /* just that.                          */
    int irr;                                    /* isr version of the int. req. reg  */
    int a2d_data;                       /* The input from the A/D converter  */
    long int tempdata;

    static int channel = 0;         /*  A/D channel to be converted      */

    dummy = (pregs < pregs) && (pmsg > pmsg);           /* avoid compiler ERRMSG*/
    dummy = (pisrblk < pisrblk);

    irr = (inp( DAS8_STATUS ) >> 4) & 0x03;         /*   get the int. req. reg. */
    das8_state = das8_state | D3_OUT_HI;   /*  set ext "in serv" flag  */
    outp( DAS8_CONTROL, das8_state );               /*  int. and clear int FF*/

    switch( irr )
    {
      case 0: printf(" *** ERR: INT SRC INDETERMINATE ***\n");
              outp( PIC, SPEC_EOI );                        /* Issue specific EOI.   */
              das8_state = das8_state & ~D3_OUT_HI;         /* clear ext isr bit */
              outp( DAS8_CONTROL, das8_state );
              return;

      case FRAME:                           /*      Frame clock          */
            if( end_frame )                         /*  If not interrupting self:*/
            {
              ++frame;                              /*  Increment frame #         */
              end_frame = NO;                       /*  Maintain"end of frame"flg*/
              das8_state = das8_state & ~D3_OUT_HI;  /* clear ext isr bit */
              outp( DAS8_CONTROL, das8_state );
              outp( START_12_BIT, 0x00 ); /*   Start 12-bit conversion.*/
            }
            else
              printf("O!");
            outp( PIC, SPEC_EOI );                  /*     Issue specific EOI.   */
            return;
     case A2D:                                      /*  An A/D-generated interrupt */
              a2d_data =  ( inp(A2D_LO_BYTE) >> 4 ) & 0x00F; /* get lo byte */
              a2d_data += ( inp(A2D_HI_BYTE) << 4 ) & 0xFF0; /* add hi byte */
              a2d_data -= 0x0800;                   /* convert offset to 2's compliment */
                                                    /* Increment A/D mux channel:     */
              das8_state = (das8_state & ~0x07) | ((channel + 1) % N_A2D_CHAN);
              outp( DAS8_CONTROL, das8_state );
```

```
            switch( channel )
            {
           case ART_P:                          /*      Arterial pressure     */
               tempdata = (long int)k_art_p * (long int)a2d_data;
                art_press = ( (int)(tempdata/(long int)100) - off_art_p);
               if( art_press <=0 )
                art_press = 0;
                if( calflag == 0 ) {
                        data[ a_p_index ][ar] = art_press;
                        data_base[ a_p_index ] = art_press;
                }
                else {
                        data[ a_p_index ][ar] = a2d_data;
                        a_p_index = ++a_p_index % MAX_MEASURED;
                        break;
                }

/*      The following code computes the new  motor velocity in rpm, given the      */
/*      voltage representing the pressure error signal.                            */

                motor_v = bp_d1[bp_ind];
                dac_v = (int)((((long int)bp_d0[bp_ind] * (long int)alpha) +
                                ((long int)minalpha * (long int)beta)) / (long int)100);

                if( dac_v < 800 ) dac_v = 800;
                if( dac_v > 4095 ) dac_v = 4095;
                data[ a_p_index ][ec] = ecg_des[(bp_ind+current_length-20) % current_length];

                bp_ind = ++bp_ind % current_length;
                tempdata = (long int)dac_v * (long int)6000;
                data[ a_p_index ][pp] = (int)(tempdata / 4095);

               a_p_index = ++a_p_index % MAX_MEASURED;

               outp(DAC0_HI,(dac_v >> 4));
               outp(DAC0_LO,(dac_v << 4));
               outp(DAC1_HI,(motor_v >> 4));
               outp(DAC1_LO,(motor_v << 4));
               break;

           case CUFF_P:
               tempdata = (long int)k_cuff_p * (long int)a2d_data;
                cuf_press = ( (int)(tempdata/(long int)100)-off_cuff_p);
               if( cuf_press <=0 )
                cuf_press = 0;
                if( calflag == 0 ) {
                        data[ c_p_index ][cu] = cuf_press;
                }
                else {
                        data[ c_p_index ][cu] = a2d_data;
                        c_p_index = ++c_p_index % MAX_MEASURED;
                        break;
                }

                c_p_index = ++c_p_index % MAX_MEASURED;
               if( sys_check == 0)
                   {
                   cuf_press = cuf_press   (10 * sys_valve);
                   if( cuf_press >= 200 )
                        {
                        ++sys_check;
```

```
                    )
                }
            else if( sys_check == 1 ) {
                cuf_press = cuf_press - (10 * sys_valve);
                if( cuf_press <= 150 ) {
                    ++sys_check;
                    das8_state = das8_state | D2_OUT_HI;
                    outp(DAS8_CONTROL,das8_state);
                }
            }
            else if( sys_check == BLEED_TIME)
                {
                das8_state = das8_state & ~D2_OUT_HI;
                outp(DAS8_CONTROL,das8_state);
                ++sys_check;
                }
            else if( sys_check > BLEED_TIME)
                {
                ++sys_check;
                cuf_press = cuf_press -  (dias_valve * 10);
                if( cuf_press < 0 ) sys_check = 0;
                }
            else ++sys_check;
            break;

        case VEIN_P:
            tempdata = (long int)k_vein_p * (long int)a2d_data;
             vein_press = ( (int)(tempdata/(long int)100) - off_vein_p);
            if( vein_press <=0 )
              vein_press = 0;
             if( calflag == 0 ) {
                    data[ v_p_index ][ve] = vein_press;
             }
             else {
                    data[ v_p_index ][ve] = a2d_data;
             }
             v_p_index = ++v_p_index % MAX_MEASURED;
             break;

        case CORE_T:
            tempdata = (long int)k_core_t * (long int)a2d_data;
             core_temp = ( (int)(tempdata/(long int)100) - off_core_t);
            data[ c_t_index ][co] = core_temp;
            c_t_index = ++c_t_index % MAX_MEASURED;
            break;

        case SKIN_T:
            tempdata = (long int)k_skin_t * (long int)a2d_data;
             skin_temp = ( (int)(tempdata/(long int)100) - off_skin_t);
            data[ s_t_index ][sk] = skin_temp;
            s_t_index = ++s_t_index % MAX_MEASURED;
            break;

    default: printf( " CHAN ERR \n" );
            break;
          }

          das8_state = das8_state & ~D3_OUT_HI;
          outp( DAS8_CONTROL, das8_state );

          channel = ++channel;
```

```
            if( channel >= N_A2D_CHAN )          /* IF all A/D channels scanned: */
            {
               channel = 0;                  /* Reset channel counter  */
               end_frame = YES;              /* Flag end of frame      */
            last_measured = ++last_measured % MAX_MEASURED;

            }
            else {

              outp( START_12_BIT, 0x00 );        /*   Start 12-bit conversion.*/
            }
      }

      outp( PIC, SPEC_EOI );                     /*   Issue specific EOI      */
      return;
}
```